Министерство науки и высшего образования Российской Федерации

ФГБОУ ВО «Кубанский государственный технологический университет» (ФГБОУ ВО «КубГТУ»)

Институт Компьютерных систем и информационной безопасности

Кафедра Информационных систем и программирования

Направление подготовки/специальность 09.03.04 Программная инженерия

(код и наименование направления подготовки/специальности)

Профиль/специализация Проектирование и разработка программного обеспечения

(наименование профиля/специализации)

Курсовой проект

по дисциплине Технологии разработки программного обеспечения

на тему «Банкомат»

Выполнил(-а) студент(-ка) Козляк Р.В. курса 2 группы 19-КБ-ПР2

Допущена к защите 15.12.2020

(дата)

Руководитель (нормоконтролер) работы Попова О.Б. Фамилия И.О

Защищена 17.12.2020 Оценка отл

(дата)

Члены комиссии: Кушнир Н.В.

Тотухов К.Е.

(должность, подпись, дата, расшифровка подписи)

Краснодар

2020 г.

Институт Компьютерных систем и информационной безопасности

Кафедра Информационных систем и программирования

Направление подготовки/специальность 09.03.04 Программная инженерия

(код и наименование напрявления подготовки/специальности)

Профиль/специализация Проектирование и разработка программного обеспечения

(наименование профиля/специализации)

Утверждаю

Зав.Кафедрой

М.В. Янковой

«23» сентября 2020 г.

Задания

на курсовой проект

Студенту Козляку Руслану Владимировичу курса 2 группы 19-КБ-ПР2

Тема работы: «Банкомат»

(утверждена указанием директора института №21-кот 13.09 2020 г.)

План работы:

1.Изучение предметной области

2.Проектирование

3.Описание реализованных диаграмм

Обьем работы:

а) пояснительная записка 41 с.

б) иллюстративная часть 24 лист(-ов)

Рекомендуемая литература:

1.Роберт А. Максимчук. UML для простых смертных
2.Йордон, Эдвард. Объектно-ориентированный анализ и проектирование систем / Эдвард Йордон , Карл Аргила. - М.: ЛОРИ, 2014. - 264 с.

Срок выполнения: с «08» сентября по «19» декабря 2020 г.

Срок защиты: «26» декабря 2020 г.

Дата выдачи задания: «08» сентября 2020 г.

Дата сдачи работы на кафедру: «26» декабря 2020 г.

Руководитель работы доц. Попова О.Б.

(должность, подпись)

Задание принял студент Козляк Р.В.

Реферат

Пояснительная записка курсового проекта: 41 с. , 24 рис. , 2 табл. , 8 источников , 2 прил.

ПРОЕКТИРОВАНИЕ, МОДЕЛЬ, КЛАСС, ВРЕМЕННАЯ ДИАГРАММА, СЕТЕВАЯ ДИАГРАММА, UML ДИАГРАММА, ДИАГРАММА ГАНТА, IDEF3, IDEF0, IDEF1X

Объектом исследования является программное обеспечение банкомата, в функционал которого входят такие возможности как: прием пластиковых карт, распознавание их принадлежности и проверки введенного пин кода, блокировка карт в случае трех попыток не верного ввода пин кода, вывод информации о счете клиента, выдача наличных по запросу в случае если их достаточно как на счету клиента так и в самом терминале, загрузка средств оператором терминала.

Цель работы состоит в разработке проекта программного обеспечения «Банкомат» с использованием диаграмм разного вида, в полной мере описывающих как внутреннее устройство исследуемой системы, так и всевозможные взаимодействия между её компонентами.

В результате были получены диаграммы, обладающие исчерпывающей информацией о программном обеспечение банкомата. К ним относятся: диаграмма Ганта, сетевая диаграмма этапов, временная диаграмма распределения работников по этапам, IDEF0-диаграмма, IDEF1X-диаграмма, IDEF3-диаграмма, все UML диаграммы: диаграмма классов, диаграмма компонентов и размещений, диаграмма пакетов, диаграмма последовательности, диаграмма кооперации, диаграмма состояний, диаграмма вариантов использования.

Содержание

Введение

В настоящее время банкоматы служат одним из основных средств выдачи наличных денежных средств. К числу достоинств, послуживших причиной такого глобального распространения технологии, можно отнести простоту обращения, надёжность, разгрузку банковских филиалов от больших очередей из желающих списать со своего счета наличные денежные средства. Применение банковских терминалов повсеместно стало неотъемлемой частью человеческой жизни.

Для разработки и отладки программного обеспечения банковского терминала в данном проекте используется симулятор банкомата, позволяющий в полной мере изучить взаимодействие всех системообразующих компонентов и провести полноценное их тестирование.

Вся работа разделена следующим образом:

Козляк Р.В. выполняет построение диаграмм Ганта, IDEF1x, IDEF0 и IDEF3, а так же перерасчёт трудозатрат.

Отришко А.А. выполняет построение всех UML диаграмм, а так же реализацию нового функционала банкомата, его тестирование и описание в руководстве пользователя.

1. Основная часть

1.1 Формулировка задачи

Задачей данного курсового проекта является разработка модели программного обеспечения встроенного микропроцессора для банкомата. Банкомат должен состоять из следующих компонентов:

– цифрового дисплея для визуального отображения, вводимой информации;

– панель управления;

– приемник кредитных карт;

– хранилище денежных средств;

– лотка для выдачи денежных средств;

– хранилище конфискованных кредитных карт;

– встроенный принтер для выдачи справок о проведенных операциях;

Панель управления представляет собой обычную клавиатуру с урезанным функционалом (доступен ввод исключительно представленной цифрами информации). Ввиду невозможности реализации физического приемника кредитных карт реализован виртуальный (прием кредитной карты производится посредством считывания введенных пользователем данных карты по запросу системы). Хранилище денежных средств по той же причине представлено в виде виртуального и хранится вне памяти системы и считывается при каждом ее запуске. Лоток для выдачи денежных средств так же представлен в виртуальном виде и реализован через списание средств с счета терминала и клиента. Хранилище конфискованных кредитных карт реализовано в виде реестра, куда заносятся данные карт, число неудачных попыток авторизации которых превысило 3. Встроенный принтер реализован через вывод справки о произведенной операции.

1.2 Диаграмма Ганта

Диаграмма Ганта — «это популярный тип столбчатых диаграмм (гистограмм), который используется для иллюстрации плана, графика работ по какому-либо проекту. Является одним из методов планирования проектов. Придумал американский инженер Генри Гант (Henry Gantt). Выглядит это как горизонтальные полосы, расположенные между двумя осями: списком задач по вертикали и датами по горизонтали.

На диаграмме видны не только сами задачи, но и их последовательность. Это позволяет ни о чём не забыть и делать всё своевременно.

Ключевым понятием диаграммы Ганта является «веха» — метка значимого момента в ходе выполнения работ, общая граница двух или более задач. Вехи позволяют наглядно отобразить необходимость синхронизации, последовательности в выполнении различных работ. Вехи, как и другие границы на диаграмме, не являются календарными датами. Сдвиг вехи приводит к сдвигу всего проекта. Поэтому диаграмма Ганта не является, строго говоря, графиком работ. Кроме того, диаграмма Ганта не отображает значимости или ресурсоемкости работ, не отображает сущности работ (области действия). Для крупных проектов диаграмма Ганта становится чрезмерно тяжеловесной и теряет всякую наглядность.»

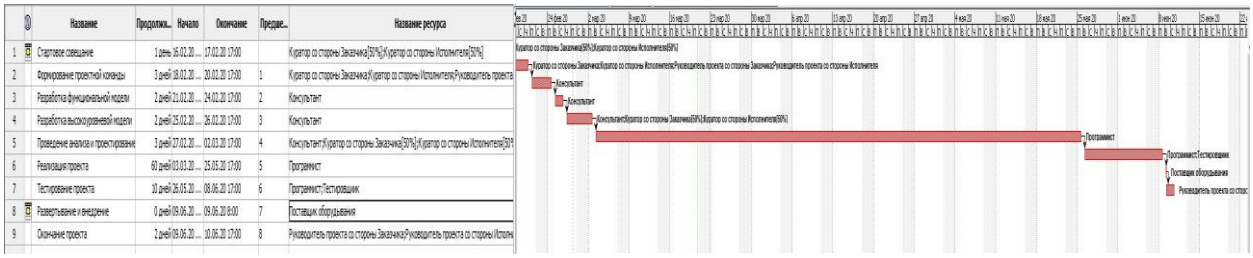


Рисунок 1 – Диаграмма Ганта до изменений

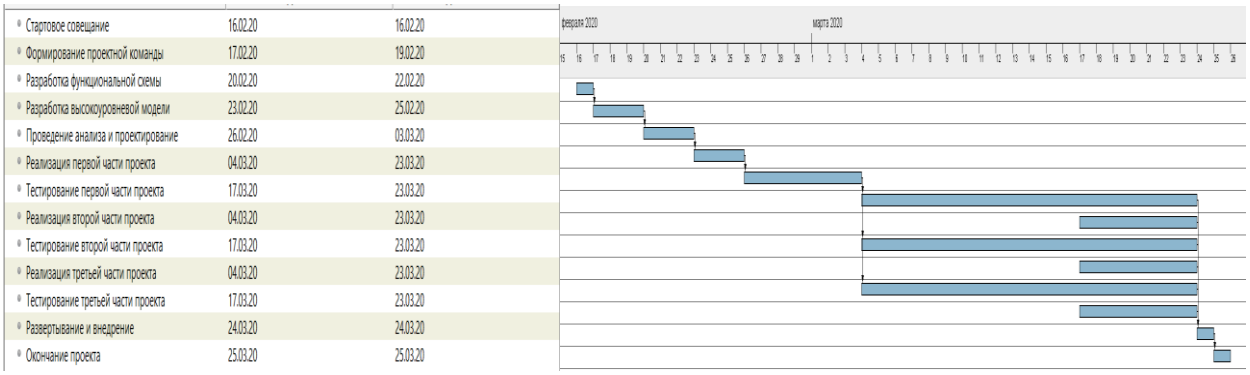


Рисунок 2 – Диаграмма Ганта после изменений

Благодаря распараллеливанию процесса разработки и тестирования проекта удалось получить выигрыш во времени. Против изначальных 70 дней (60 дней реализации проекта + 10 дней тестирования) мы получаем 20 дней после распараллеливания. Так же это позволяет сократить расходы на зарплаты команды разработчиков и тестировщиков.

1.3 Сетевая диаграмма

Сетевая диаграмма отображает зависимости между различными этапами проекта. Основной целью её использования является эффективное планирование и управление работами и ресурсами проекта. Если для создания сетевой диаграммы используются программные средства поддержки управления проектом, каждый этап должен заканчиваться контрольной отметкой. Очередной этап может начаться только тогда, когда будет получена контрольная отметка (которая может зависеть от нескольких предшествующих этапов). Любой этап не может начаться, пока не выполнены все этапы на всех путях, ведущих от начала проекта к данному этапу. Минимальное время выполнения всего проекта можно рассчитать, просуммировав в сетевой диаграмме длительности этапов на самом длинном пути от начала проекта до его окончания. Таким образом, общая продолжительность реализации проекта зависит от этапов работ, находящихся на критическом пути. Любая задержка в завершении любого этапа на критическом пути приведет к задержке всего проекта. Задержка в завершении

этапов, не входящих в критический путь, не влияет на продолжительность всего проекта до тех пор, пока суммарная длительность этих этапов на каком-нибудь пути не превысит продолжительности работ на критическом пути.

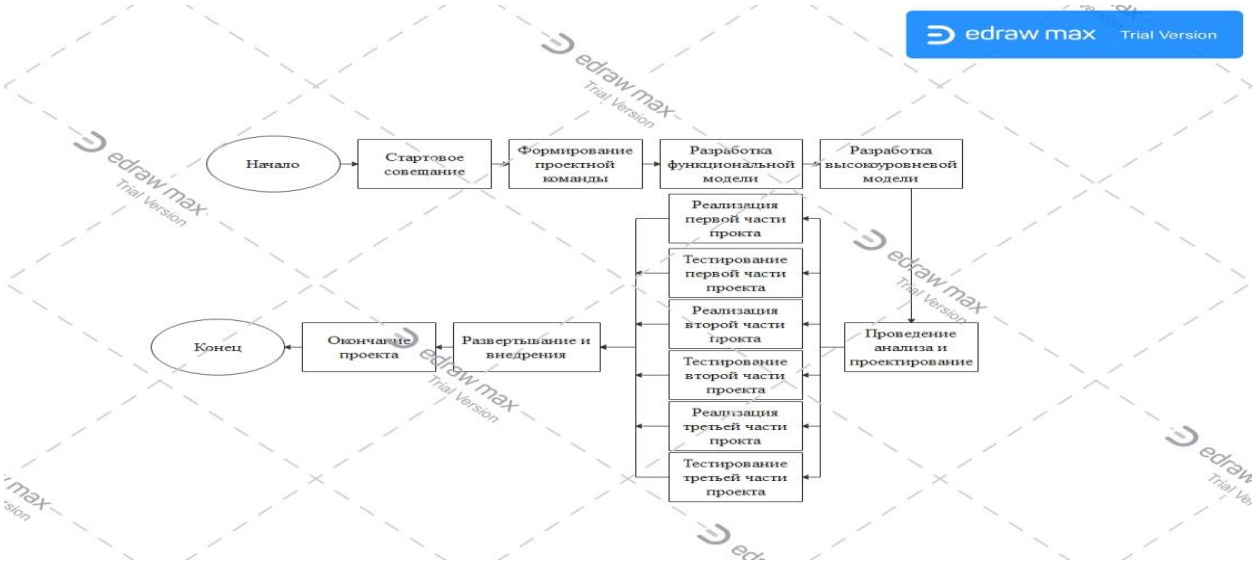


Рисунок 3 – Сетевая диаграмма

1.4 Создание модели As–Is в стандарте IDEF0

Чтобы оценить возможности, разрабатываемой системы необходимо построить её базовую модель, которую можно представить в виде диаграммы As-Is.

Диаграмма As-Is – это функциональная модель системы «как есть», позволяющая узнать где находятся слабые места, в чём будут состоять преимущества и недостатки, протекающих в ней. Применение данной модели позволит чётко зафиксировать какие информационные объекты принимают участие в жизненном цикле системы, какая информация будет поступать на вход и что будет получаться на выходе. Модель As-Is, строится с использованием нотации IDEF0.

IDEF0 – это графическая нотация, предназначенная для описания бизнес-процессов. Система, описываемая в данной нотации, проходит через декомпозицию или, иными словами, разбиение на взаимосвязанные функции. Для каждой функции существует правило сторон:

– стрелкой слева обозначаются входные данные;

– стрелкой сверху – управление;

– стрелкой справа – выходные данные;

– стрелкой снизу – механизм.

Учитывая всё вышеперечисленное на рисунке 1 была составлена модель As-Is проекта «Банкомат». Полученная модель может быть представлена в более подробном виде путём разбиения на большее количество составных элементов.

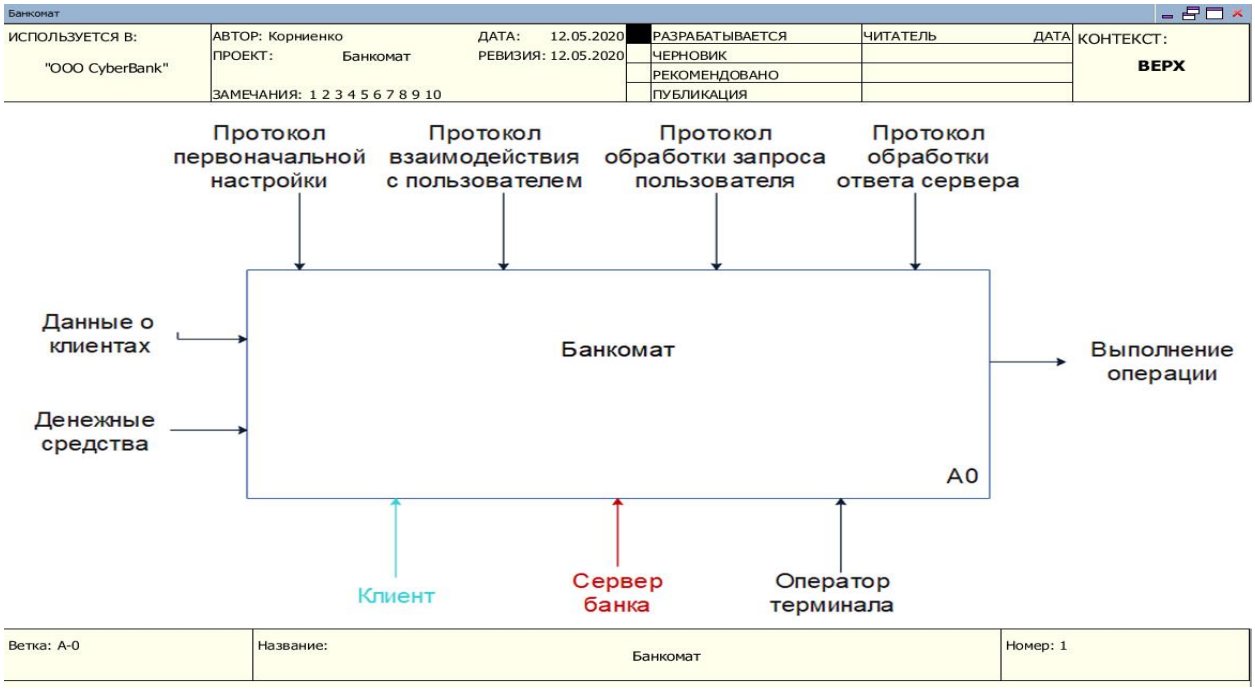


Рисунок 4 – Модель As-Is проекта «Банкомат»

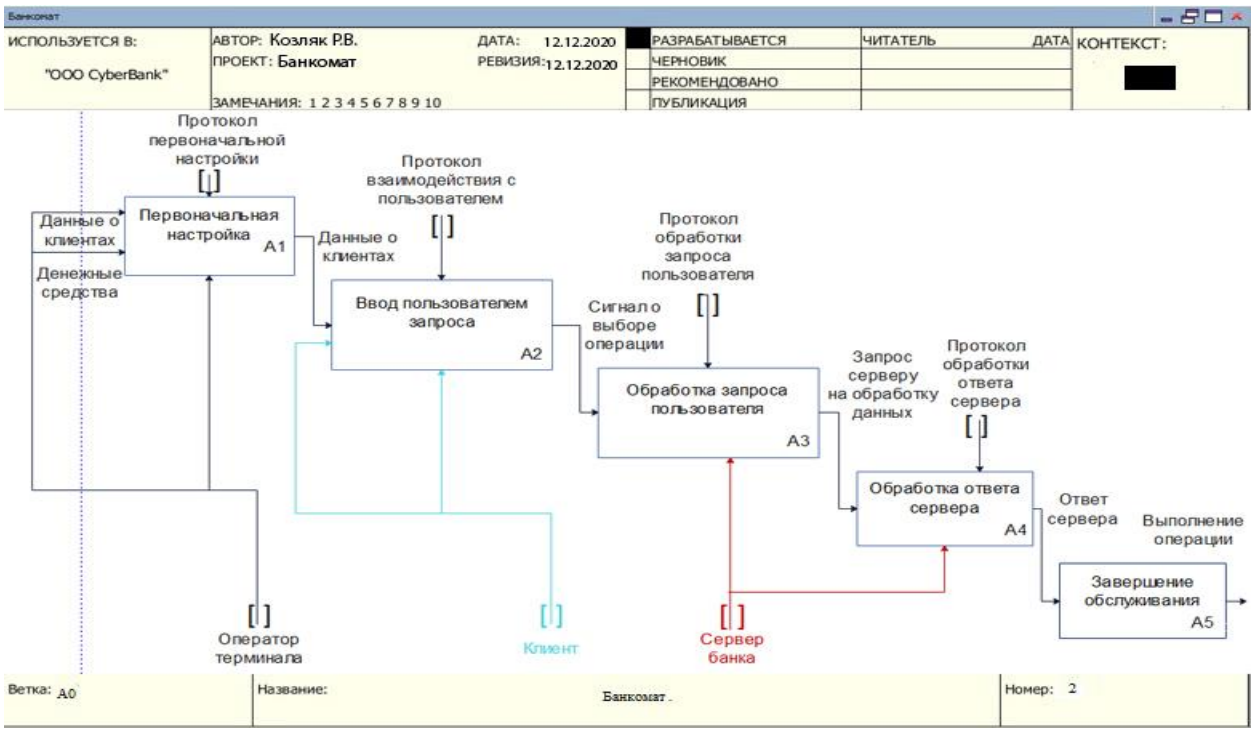


Рисунок 5 – Декомпозиция проекта «Банкомат»

Ниже представлены декомпозиции составляющих элементов разработки конечного продукта

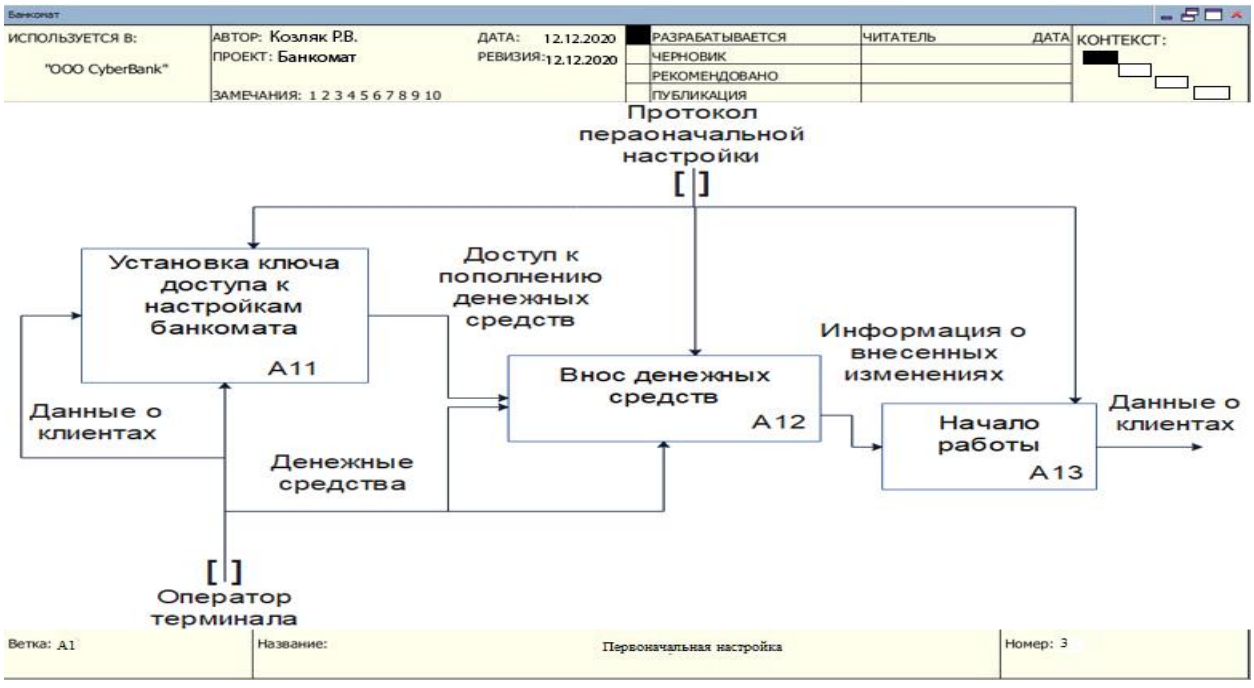


Рисунок 6 – Декомпозиция первоначальной настройки

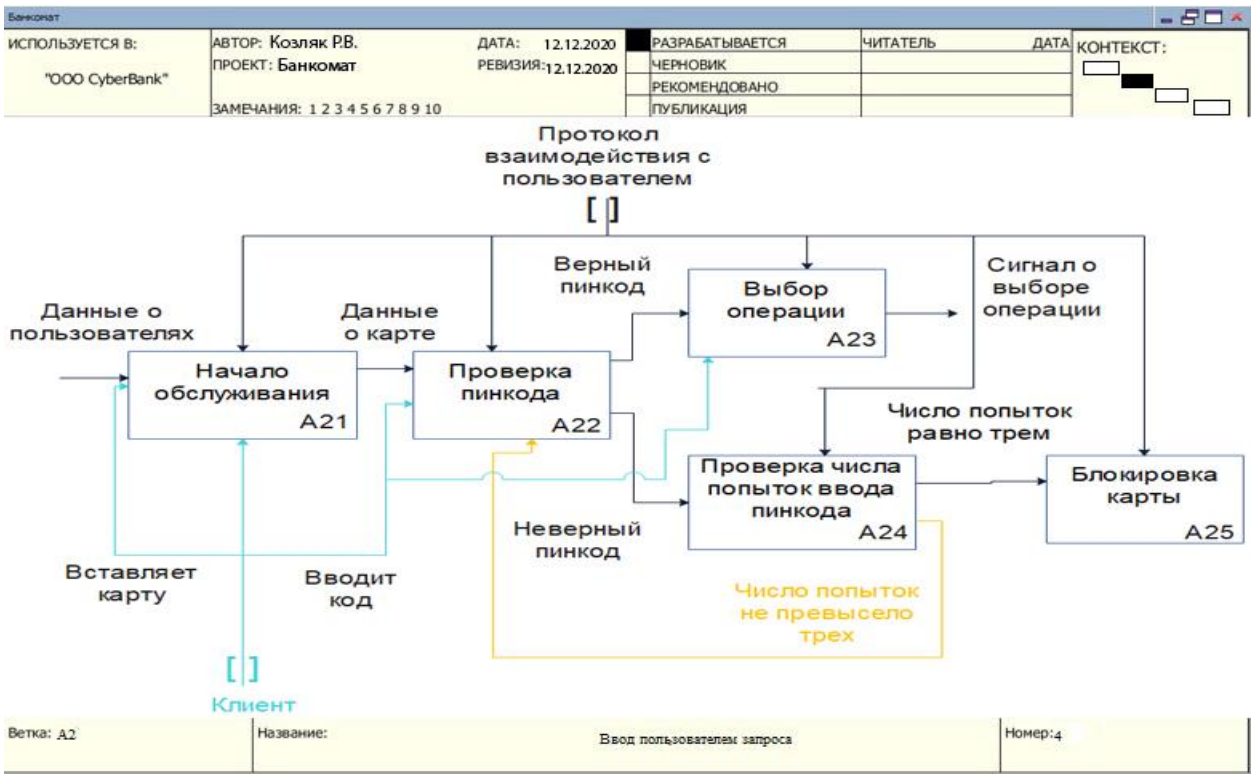


Рисунок 7 – Декомпозиция ввода пользователем запроса

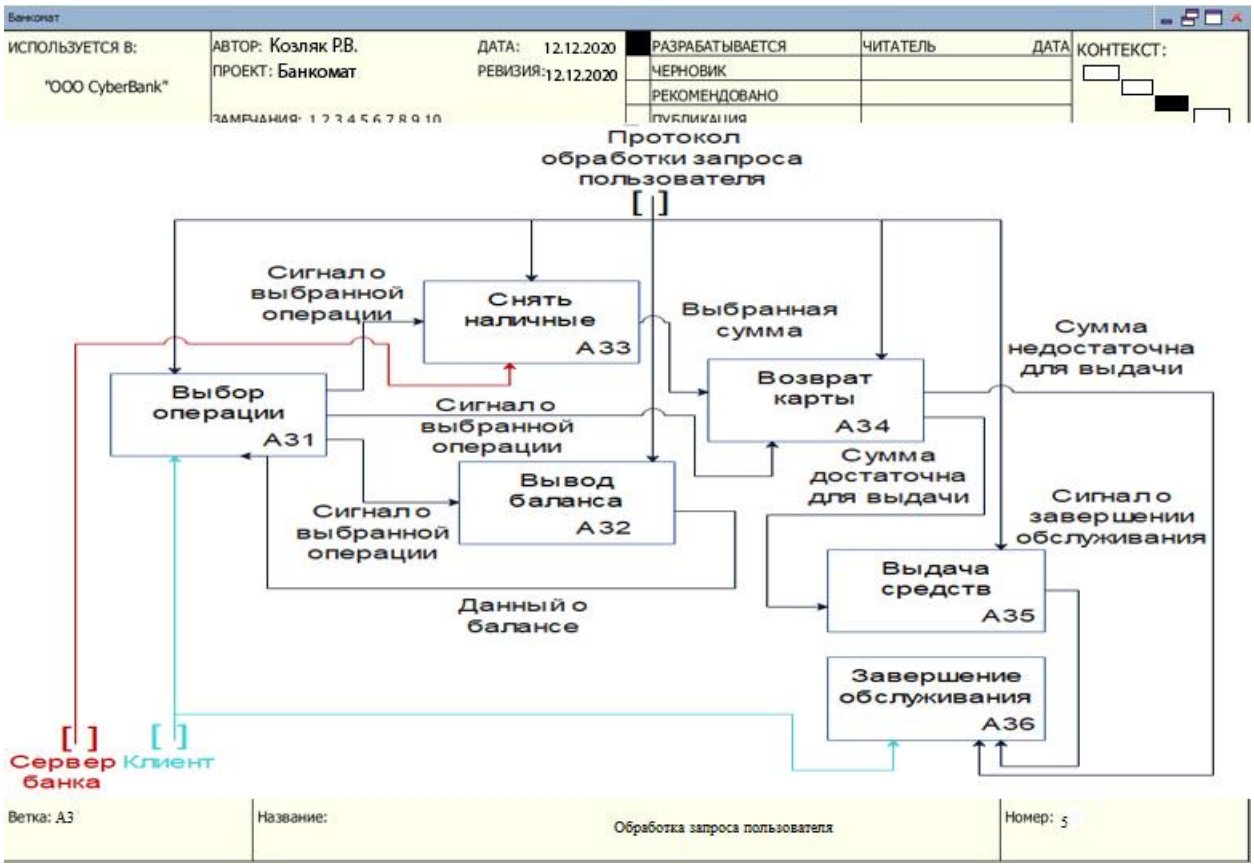


Рисунок 8 – Декомпозиция обработки запроса пользователя

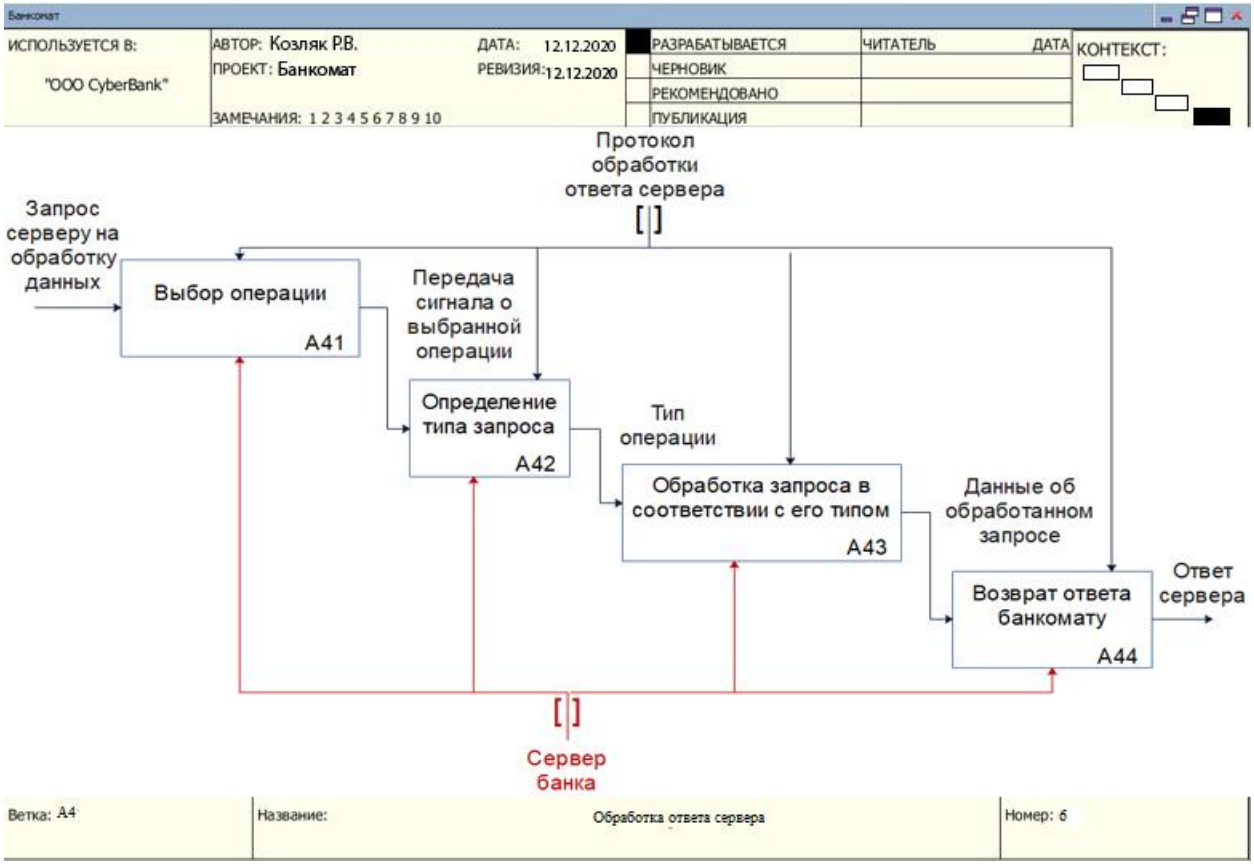


Рисунок 9 – Декомпозиция ответа сервера

Входными данными для данной системы в случае работы оператора с ней являются данные о картах клиентов, внос денежных средств, установка ключа доступа для дальнейшего обслуживания терминала.

Входными данными для данной системы в случае работы клиента с ней являются карта, пинкод, действия в меню обслуживания, действия в меню печати справки.

Механизмом реализации работы системы являются оператор терминала и клиент.

Результатом деятельности системы является выполнение запрошенных клиентом операций.

1.5 Описание методологии IDEF3

Для описания логики взаимодействия информационных потоков наиболее подходит IDEF3, называемая также workflow diagramming – методологией моделирования, использующая графическое описание

информационных потоков, взаимоотношений между процессами обработки информации и объектов, являющихся частью этих процессов. IDEF3 – это метод, имеющий основной целью дать возможность аналитикам описать ситуацию, когда процессы выполняются в определенной последовательности, а также описать объекты, участвующие совместно в одном процессе. IDEF3 дополняет IDEF0 и содержит все необходимое для построения моделей, которые в дальнейшем могут быть использованы для имитационного анализа. Основные описательные блоки диаграммы IDEF3:

1. Работы – являются центральными компонентами модели, изображаются прямоугольниками с прямыми углами и имеют имя, обозначающее процесс действия.

2. Связи – показывают взаимоотношение работ.

3. Перекрестки. Окончание одной работы может служить сигналом к началу нескольких работ, или же одна работа для своего запуска может ожидать окончание нескольких работ. Перекрестки используются для отображения логики взаимодействия стрелок при слиянии и разветвлении или для отображения множества событий, которые могут или должны быть завершены перед началом следующей работы.

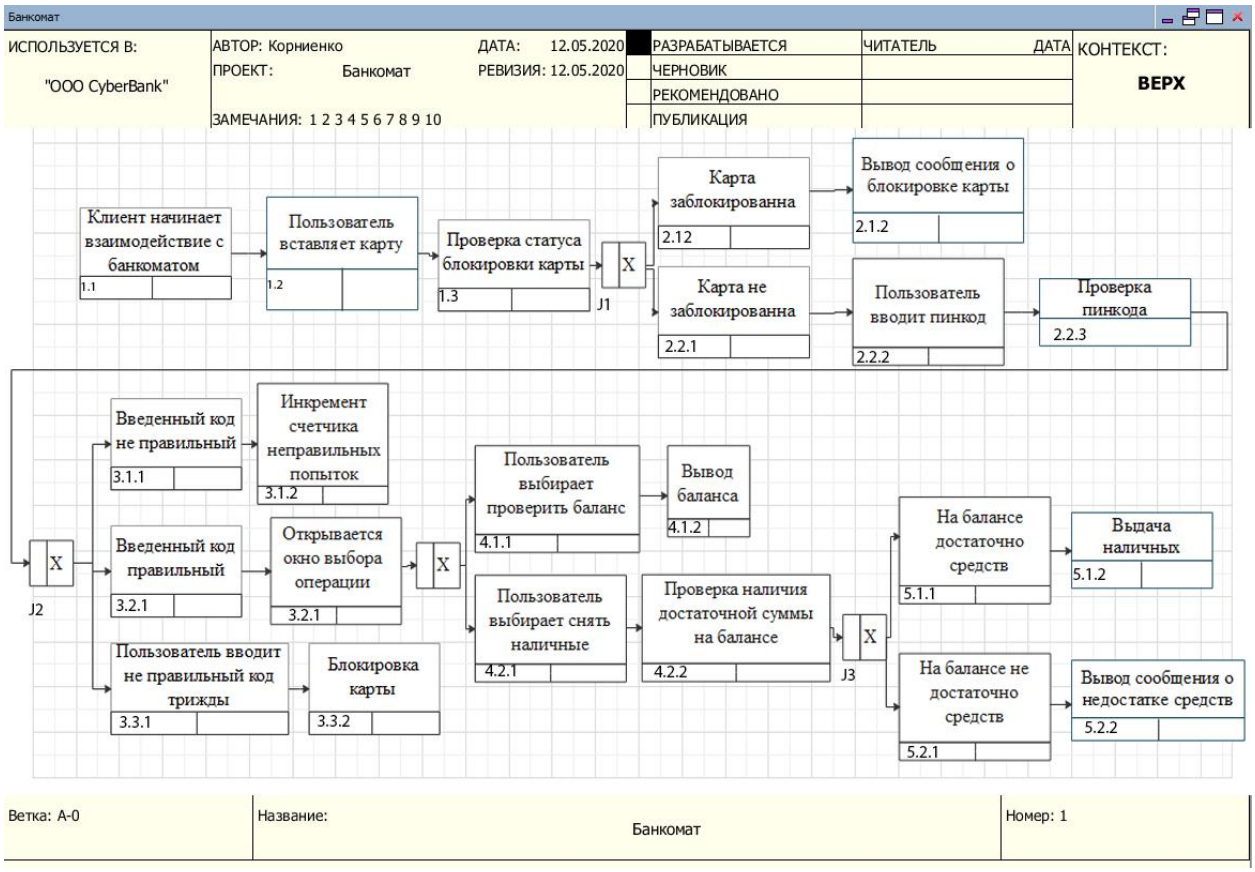


Рисунок 10 – IDEF3 диаграмма

1.6 Методология IDEFX1

Методология моделирования IDEF1X предназначена для описания данных. Чаще всего такая методология используется для описания данных в целях последующей автоматизации их обработки с помощью систем управления базами данных. Основные элементы модели IDEF1X:

1. Сущности – это множество объектов, обладающих общими характеристиками.

2. Атрибуты – характеристики сущности.

3. Отношения – это связи между двумя и более сущностями.

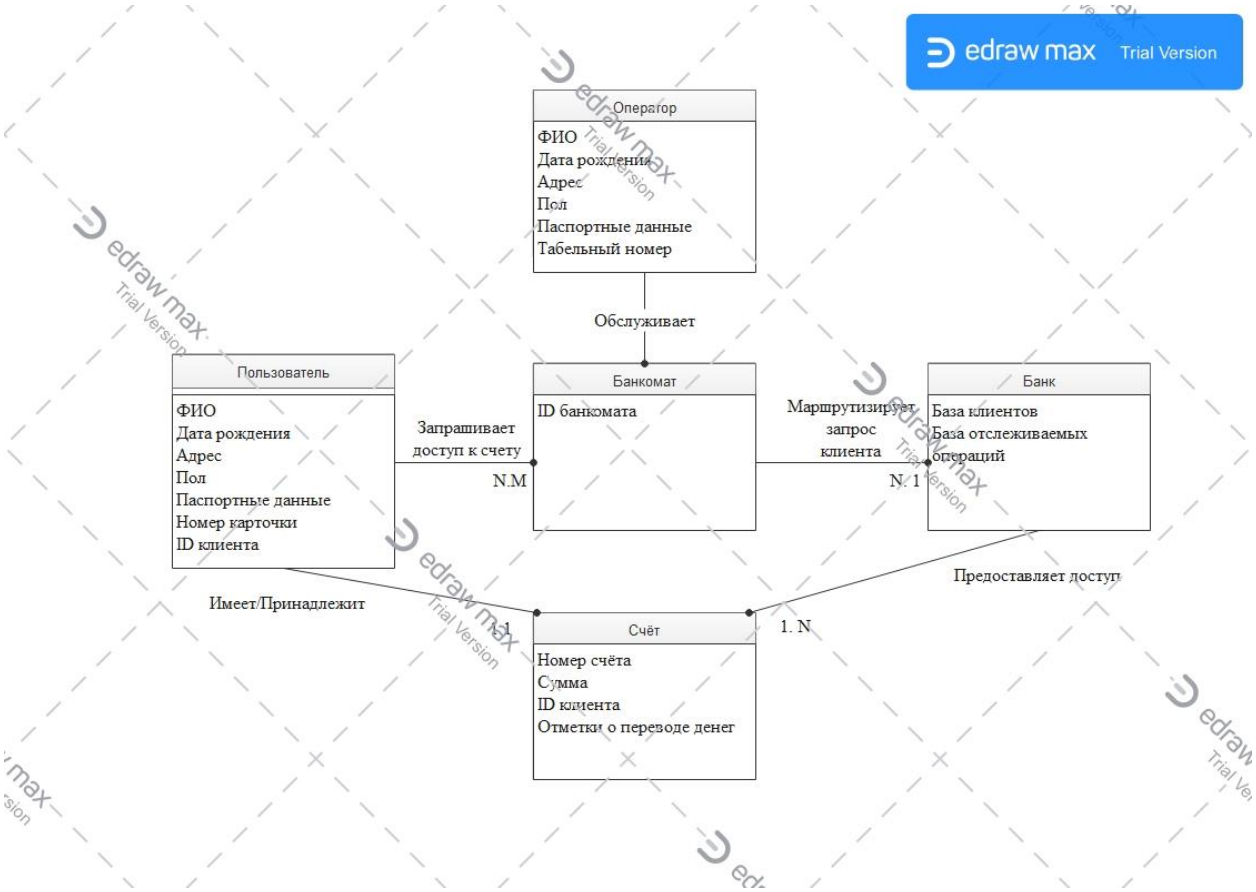


Рисунок 11 – IDEF1X диаграмма

1.7 Расчет трудоемкости методом функциональных точек

Функционально-ориентированные метрики косвенно измеряют программный продукт и процесс его разработки. Вместо подсчета LOC-оценки при этом рассматривается не размер, а функциональность или полезность продукта. В качестве количественной характеристики применяется понятие количества функциональных точек FP.

Для расчета количества функциональных точек используется пять информационных характеристик:

1) количество внешних вводов;

2) количество внешних выводов;

3) количество внешних запросов;

4) количество внутренних логических файлов;

5) количество внешних интерфейсных файлов.

Для каждого бизнес-процесса каждой выделенной характеристике ставится в соответствие сложность. Для этого характеристике назначается низкий, средний или высокий ранг, а затем формируется числовая оценка ранга. Для внешних вводов, выводов и запросов ранжирование основано на количестве ссылок на файлы и количестве элементов данных a_{ij} а где i — номер строки, соответствующей количеству внешних вводов, выводов или запросов; j — номер, соответствующий количеству элементов данных.

W=0.65+(0.01*30)

Уточненное количество функциональных точек

R(F)=142*0.95=134,9

Размерность ПО для C#

R(Loc)=134,9*5,4=728,46

Расчёты трудозатрат показывают эффективность распараллеливания этапа разработки проекта.

Таблица 1 – Расчет функциональных точек

Наименование функции	Количество функциональных точек, соответствующее категории функции	Количество функциональных точек
1. Определение количества выводов	12*4	48
2. Определение количества вводов	3*4	12
3. Определение количества опросов вывода	2*4	8
4. Определение количества опросов ввода	2*4	8
5. Определение количества файлов	3*7	21
6. Определение количества интерфейсов	5*7	35
Общее количество функциональных точек		142

Расчет происходил следующим образом:

1. При использовании для получения FP-метрик моделей Idef0 и Idef1x выводы можно определять на основе стрелок, исходящих из рассматриваемого процесса модели Idef0 и соответствующих им сущностей модели Idef1x.

2. При использовании для получения FP-метрик моделей Idef0 и Idef1x вводы можно определять на основе стрелок, входящих в рассматриваемый процесс модели Idef0 и соответствующих им сущностей модели Idef1x.

3. Под запросами при расчете FP-оценок следует понимать диалоговый ввод/вывод, который немедленно приводит к немедленному программному ответу. 4. Количество внешних файлов с данными.

5. Под интерфейсами следует понимать структуры данных, получаемых из внешних программных систем и структуры данных, передаваемые во внешние программные системы. Не стоит забывать и о коэффициентах сложности, которые прямо влияют на расчет количества функциональных точек (см таблицу 1).

1.8 UML

UML (англ. Unified Modeling Language — «унифицированный язык моделирования) — язык графического описания для объектного моделирования в области разработки программного обеспечения, для моделирования бизнес-процессов, системного проектирования и отображения организационных структур.

UML является языком широкого профиля, это — открытый стандарт, использующий графические обозначения для создания абстрактной модели системы, называемой UML-моделью. UML был создан для определения, визуализации, проектирования и документирования, в основном, программных систем. UML не является языком программирования, но на основании UML-моделей возможна генерация кода.»

В UML используются следующие виды диаграмм:

1.8.1 Диаграмма вариантов использования

Понятие варианта использования впервые ввел Ивар Якобсон. В настоящее время вариант использования превратился в основной элемент разработки и планирования проекта.

Вариант использования представляет собой последовательность действий, выполняемых системой в ответ на событие, инициируемое некоторым действующим лицом.

Действующее лицо – это роль, которую пользователь играет по отношению к системе. Действующие лица представляют собой роли, а не конкретных людей или наименования работ.

При этом на диаграмме вариантов использования существует три типа связи:

1. Связь коммуникации – это связь между вариантом использования и действующим лицом.

2. Связь включения применяется, когда имеется фрагмент поведения системы, который повторяется больше чем в одном варианте использования.

3. Связь расширения позволяет варианту использования при необходимости использовать функциональные возможности другого.

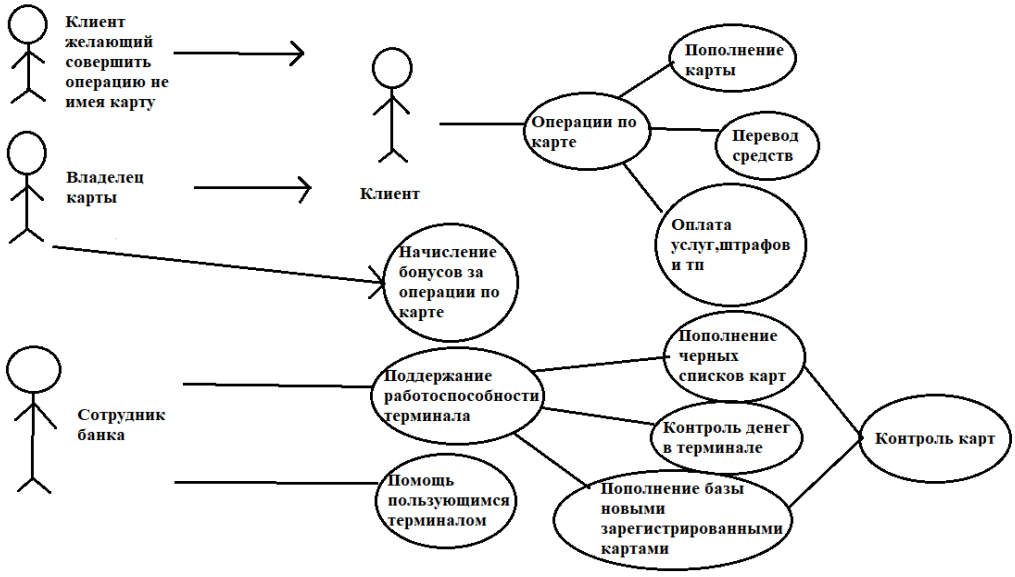


Рисунок 12 – Диаграмма вариантов использования проекта «Банкомат»

1.8.2 Диаграмма последовательности

Диаграмма последовательности отражает поток событий, происходящих в рамках варианта использования.

Действующие лица показаны в верхней части диаграммы. Стрелки соответствуют сообщениям, передаваемым между действующим лицом и объектами для выполнения требуемых функций. На диаграмме объект изображается в виде прямоугольника, от которого вниз проведена пунктирная вертикальная линия(линия жизни).

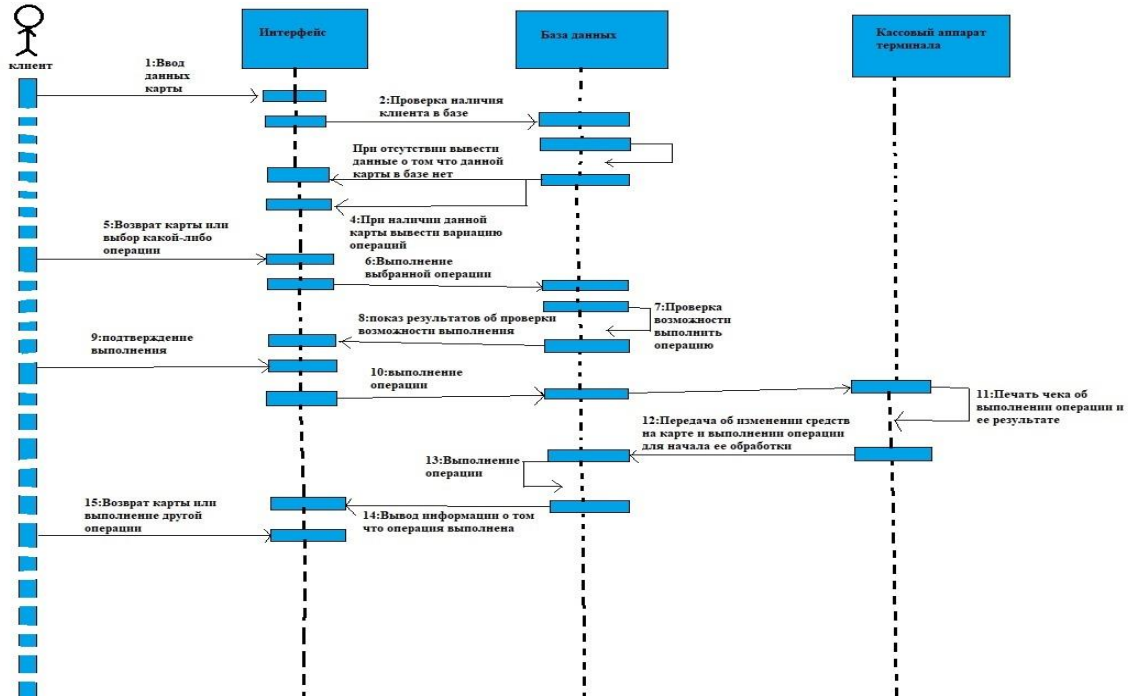


Рисунок 13 – Диаграмма последовательности проекта «Банкомат»

1.8.3 Диаграмма кооперации

На диаграмме кооперации представлена вся та информация, которая есть и на диаграмме последовательности, но она по-другому описывает поток событий. Из нее легче понять связи между объектами, но труднее понять последовательность событий.

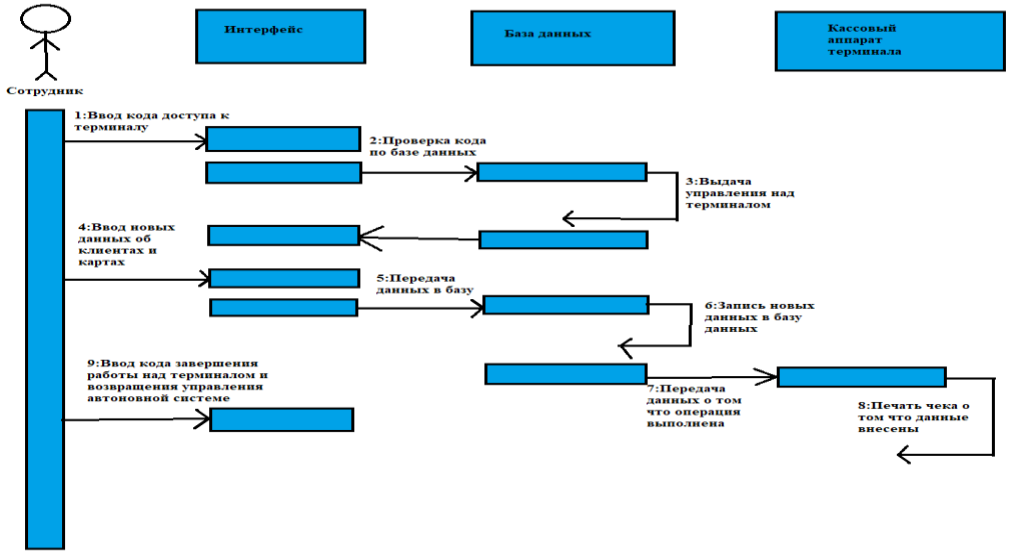


Рисунок 14 – Диаграмма кооперации проекта «Банкомат»

1.8.4 Диаграмма классов

Диаграмма классов определяет типы классов системы и различного рода статические связи, которые существуют между ними. На диаграмме классов также изображаются атрибуты классов, операции классов и ограничения, которые накладываются на связи между классами.

Диаграмма классов – это граф, узлами которого являются элементы статической структуры проекта (классы, интерфейсы), а дугами – отношение между узлами.

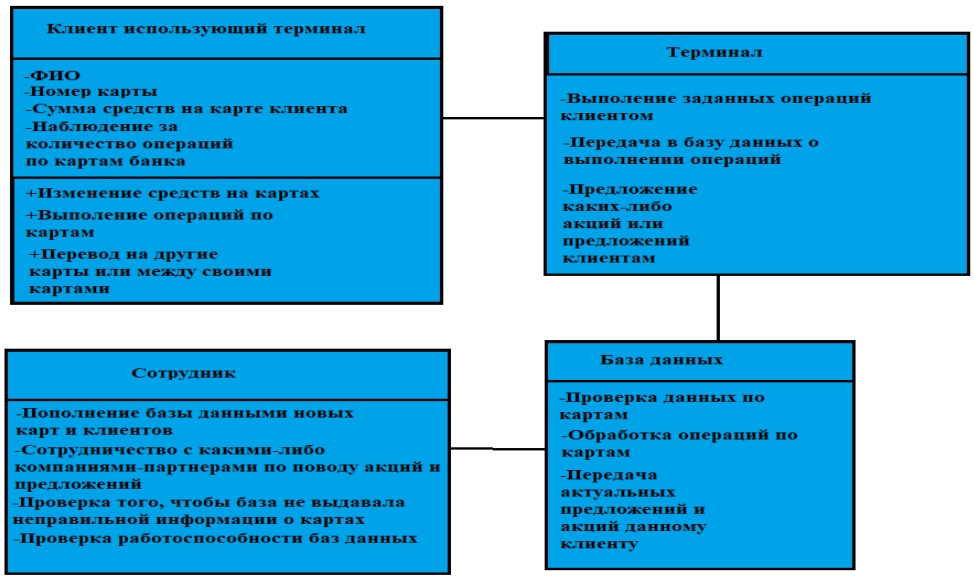


Рисунок 15 – Диаграмма классов проекта «Банкомат»

1.8.5Диаграмма состояний

Диаграммы состояний определяют все возможные состояния, в которых может находиться конкретный объект, а также процесс смены состояний объекта в результате наступления некоторых событий.

На диаграмме имеются два специальных состояния – начальное и конечное. Начальное состояние – это выделенная черная точка, конечное состояние – черная точка в белом круге. На диаграмме может отсутствовать конечное состояние или их может быть сколько угодно, но должно быть одно и только одно начальное состояние.

Диаграмма состояний состоит из:

1. Деятельность – поведение, реализуемое объектом, пока он находится в данном состоянии. Деятельность изображают внутри самого состояния, ей должно предшествовать слово do (делать) и двоеточие.

2. Входное действие – поведение, которое выполняет объект при переходе в данное состояние. Входное действие также показывают внутри состояния, ему предшествует слово entry (вход) и двоеточие.

3. Выходное действие – поведение, которое выполняет объект при выходе из данного состояния. Выходное действие изображают внутри состояния, ему предшествует слово exit (выход) и двоеточие.

4. Событие – то, что вызывает переход из одного состояния в другое. Событие размещают на диаграмме вдоль линии перехода.

5. Ограждающие условия определяют, когда переход может осуществиться, а когда нет. Ограждающие условия задавать необязательно. Ограждающие условия изображают на диаграмме вдоль линии перехода после имени события, заключая их в квадратные скобки.

6. Действие – часть перехода. Рисуют вдоль линии перехода после имени события, ему предшествует косая черта. Действие рисуют вдоль линии перехода после имени события, ему предшествует косая черта.

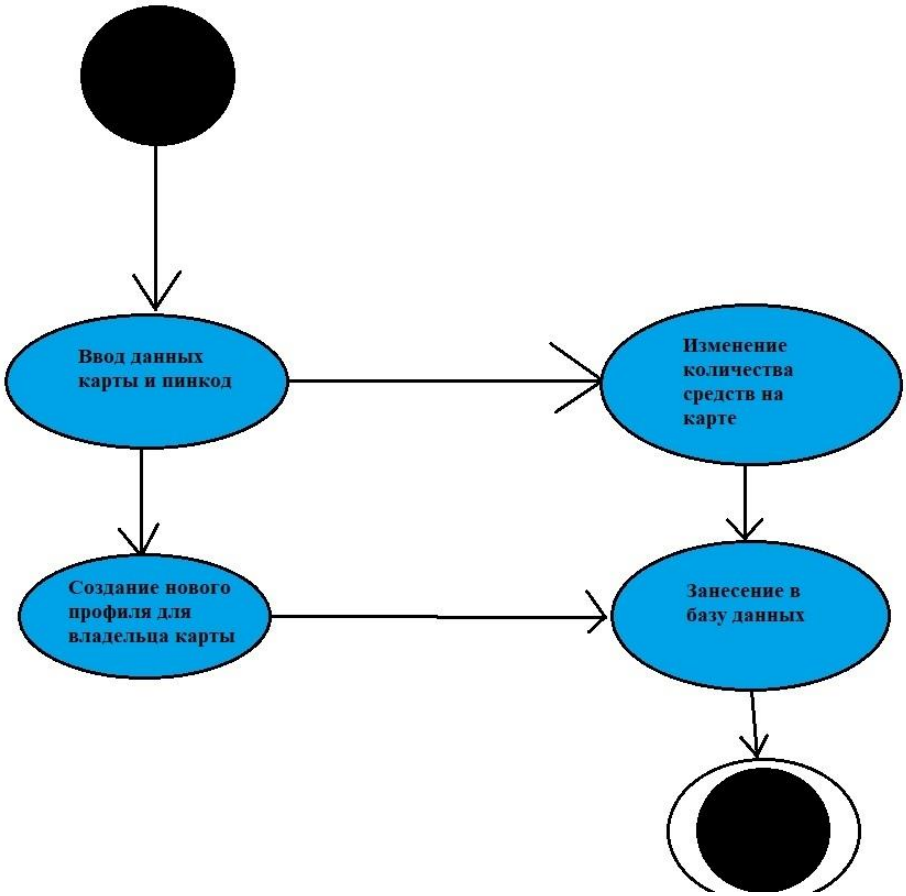


Рисунок 16 – Диаграмма состояний проекта «Банкомат»

1.8.6 Диаграмма размещения

Диаграмма размещения отражает физические взаимосвязи между программными и аппаратными компонентами системы. Она является хорошим средством для того, чтобы показать маршруты перемещения объектов и компонентов в распределенной системе.

Каждый узел на диаграмме размещения представляет собой некоторый тип вычислительного устройства – в большинстве случаев, часть аппаратуры. Эта аппаратура может быть простым устройством или датчиком, а может 31 быть и мэйнфреймом.

Диаграмма размещения показывает физическое расположение сети и местонахождение в ней различных компонентов. Диаграмма размещения используется менеджером проекта, пользователями, архитектором системы и

эксплуатационным персоналом, чтобы понять физическое размещение системы и расположение её отдельных подсистем.

И так же Диаграмма компонентов

Диаграммы компонентов показывают, как выглядит модель на физическом уровне. На них изображены компоненты программного обеспечения и связи между ними. При этом на такой диаграмме выделяют два типа компонентов:

-исполняемые компоненты;

-библиотеки кода.

Каждый класс модели (или подсистема) преобразуется в компонент исходного кода. После создания они сразу добавляются к диаграмме компонентов. Между отдельными компонентами изображают зависимости, соответствующие зависимостям на этапе компиляции или выполнения программы. Диаграммы компонентов применяются теми участниками проекта, кто отвечает за компиляцию системы. Из нее видно, в каком порядке надо компилировать компоненты, а также какие исполняемые компоненты будут созданы системой. На такой диаграмме показано соответствие классов реализованным компонентам. Она нужна там, где начинается генерация кода.

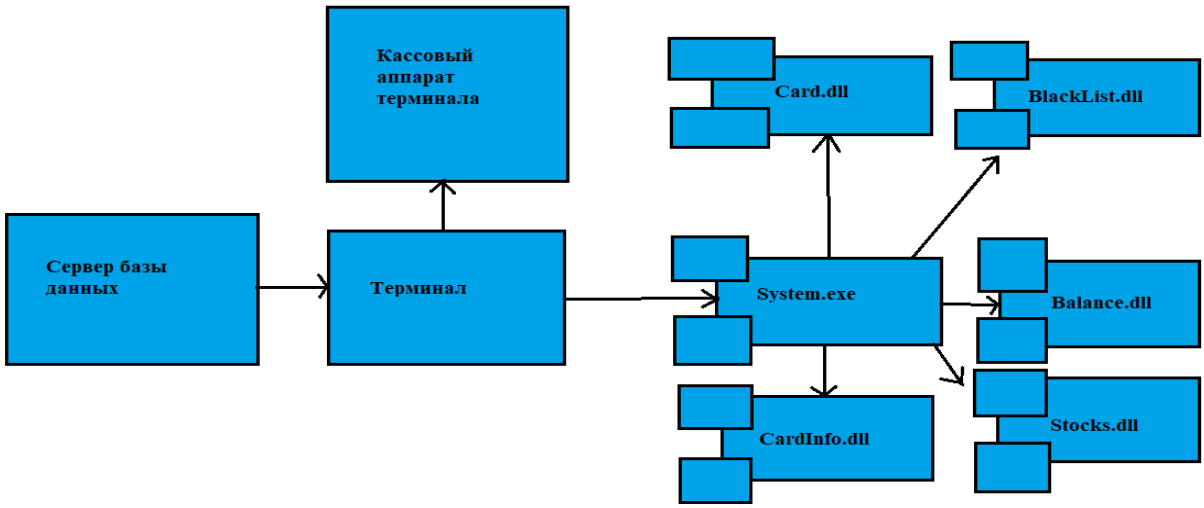


Рисунок 17 – Диаграмма компонентов проекта «Домофон»

1.9 Результаты машинного тестирования программы

На рисунке 18 представлен вид системы сразу после запуска.

Рисунок 18 – Изначальный экран запуска

На Рисунке 19 представлен встречающий экран, где необходимо ввести данные карты и пинкод от нее, для использования терминала.

Рисунок 19 – Встречающий экран в который вводятся данные карты

На рисунке 20 представлен выбор действий по карте, такие как проверка баланса, снятие наличных или завершение использования карты.

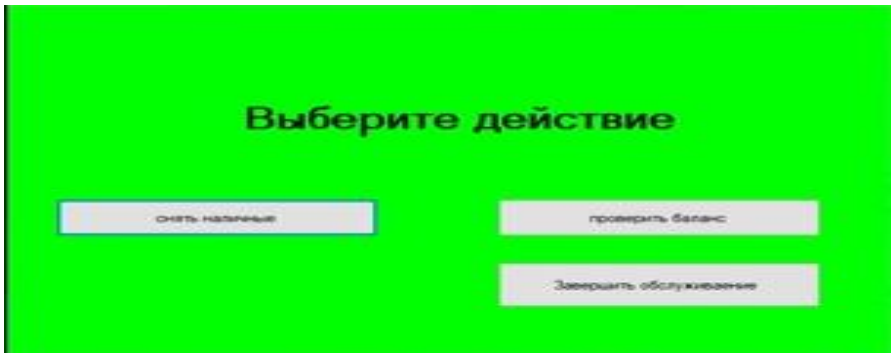


Рисунок 20 – Выбор действий по карте

На рисунке 21 показано действие которое выводится после нажатия кнопки проверить баланс. Терминал показывает средства доступные на карте в данный момент.

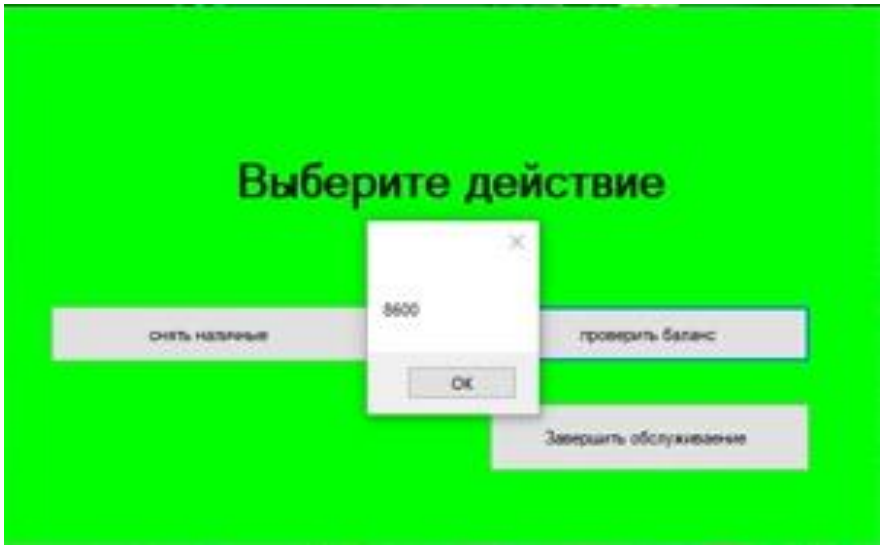


Рисунок 21 – Выбор действия проверить баланс.

На рисунке 22 показан экран который выводится после выбора действия снять наличные.

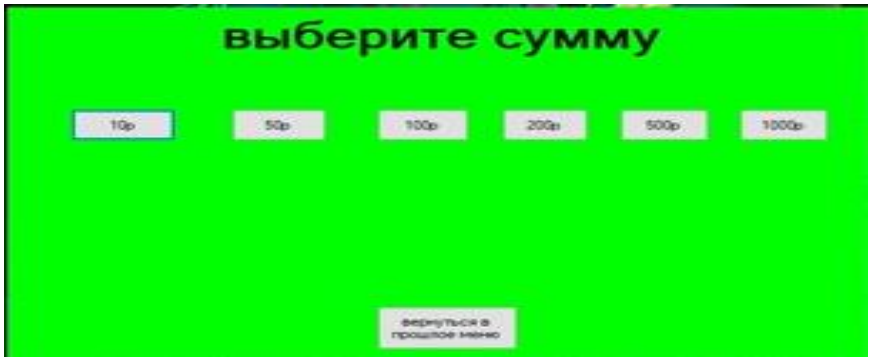


Рисунок 22 –Экран выполняющий действие снять наличные.

На рисунке 23 показан экран после выбора какой-либо суммы который спрашивает нужно ли вам распечатать чек.

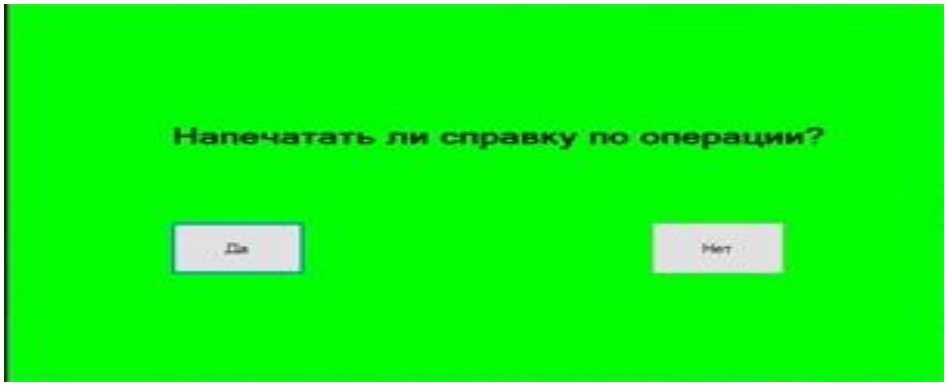


Рисунок 23 – Экран спрашивающий нужно ли напечатать чек.

На рисунке 24 показано сообщение благодарящее пользователя за использование данного банка и после этого работа терминала завершается и начинается сначала.

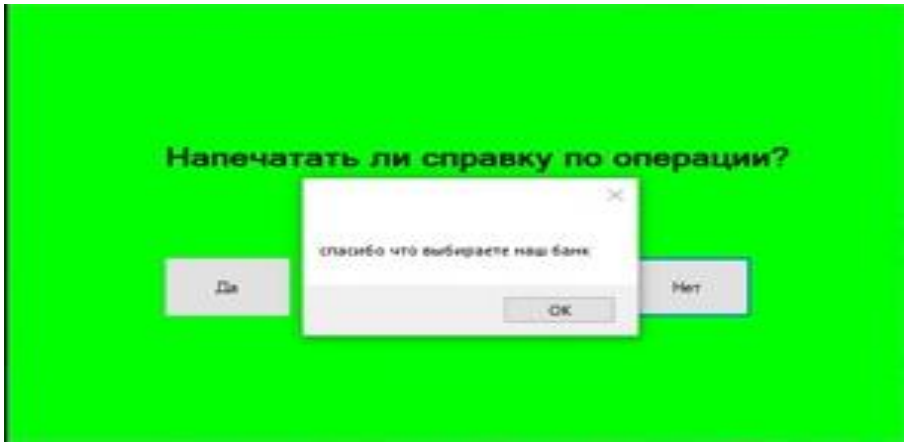


Рисунок 24 – Экран благодарности за использование данного банка

Системные требования

Таблица 2 – Системные требования программ проекта «Банкомат»

Процессор	2.5 ГГц
Оперативная память	150 Мб
Монитор	1920 x 1080
Свободное место на носителе	15 Мб
Устройства взаимодействия с пользователем	Клавиатура и мышь
Программное обеспечение	Visual Studio 2019 года последней версии

1.11 Руководство пользователя

Перед первым запуском необходимо скачать папку bank из хранилища проекта и разместить ее по адресу C:\Bank.

atmbalance - хранит баланс банкомата. (баланс не может быть больше чем 2^31)

blockedcards - хранит номера заблокированных карт.

cardinfo хранит информацию о картах клиентов в формате: строка номер карты строка пинкод строка баланс.

Для доступа в меню настроек следует в программе нажать на "банкомат" и ввести стандартный код "0000". Настройка включает в себя смену стандартного кода и внос средств в терминал.

После завершения настройки система переходит в режим обслуживания реализуя основные функции банкомата.

Для начала обслуживания клиенту необходимо ввести данные карты и ввести пинкод. В случае трех попыток ввода неверного пинкода карта блокируется системой и считается невозможной к использованию. Для восстановления карты следует удалить ее данные из «blockedcards». В случае

ввода верного пинкода клиента встречает меню обслуживания включающее в себя: Снятие наличных, отображение баланса, завершение обслуживания.

При выборе снятия наличных клиент попадает в меню выбора необходимой суммы. После выбора он получает уведомление необходимости изъять карту из банкомата без чего дальнее обслуживание не возможно. За изъятием карты следует проверка наличия достаточного колличества средств на счету клиента и в хранилище банкомата для проведения операции. В случае если на счету терминала или клиента не достаточно средств клиент получает уведомление об этом и его обслуживание завершается. В случае достатка средств на обоих счетах клиенту предлагается печать справки после чего происходит выдача средств, их списание с обоих счетов, выдача справки(опционально) и завершение обслуживания.

Заключение

В результате выполнения данного курсового проекта была спроектирована система «Банкомат» на языке высокого уровня C#, позволяющая наглядно продемонстрировать работу всех её компонентов. Полученные диаграммы обладают простой и понятной, даже для человека незнакомого с данной областью, структурой, описывающей каждый аспект системы с различных сторон.

При построении диаграмм использовались основные правила и принципы моделирования, включающие графическое представление объектов и связей между ними, иерархическое построение, а также названия, отражающие назначение той или иной сущности, или взаимодействия.

Благодаря детальному разбору проекта при помощи диаграмм проектирования, полученных в процессе разработки, можно с уверенностью сказать, что полученная система «Банкомат» полностью соответствует современным стандартам безопасности и способна выполнять различные взаимодействия, как для коммуникации с человеком, так и для повышения уровня защищённости.

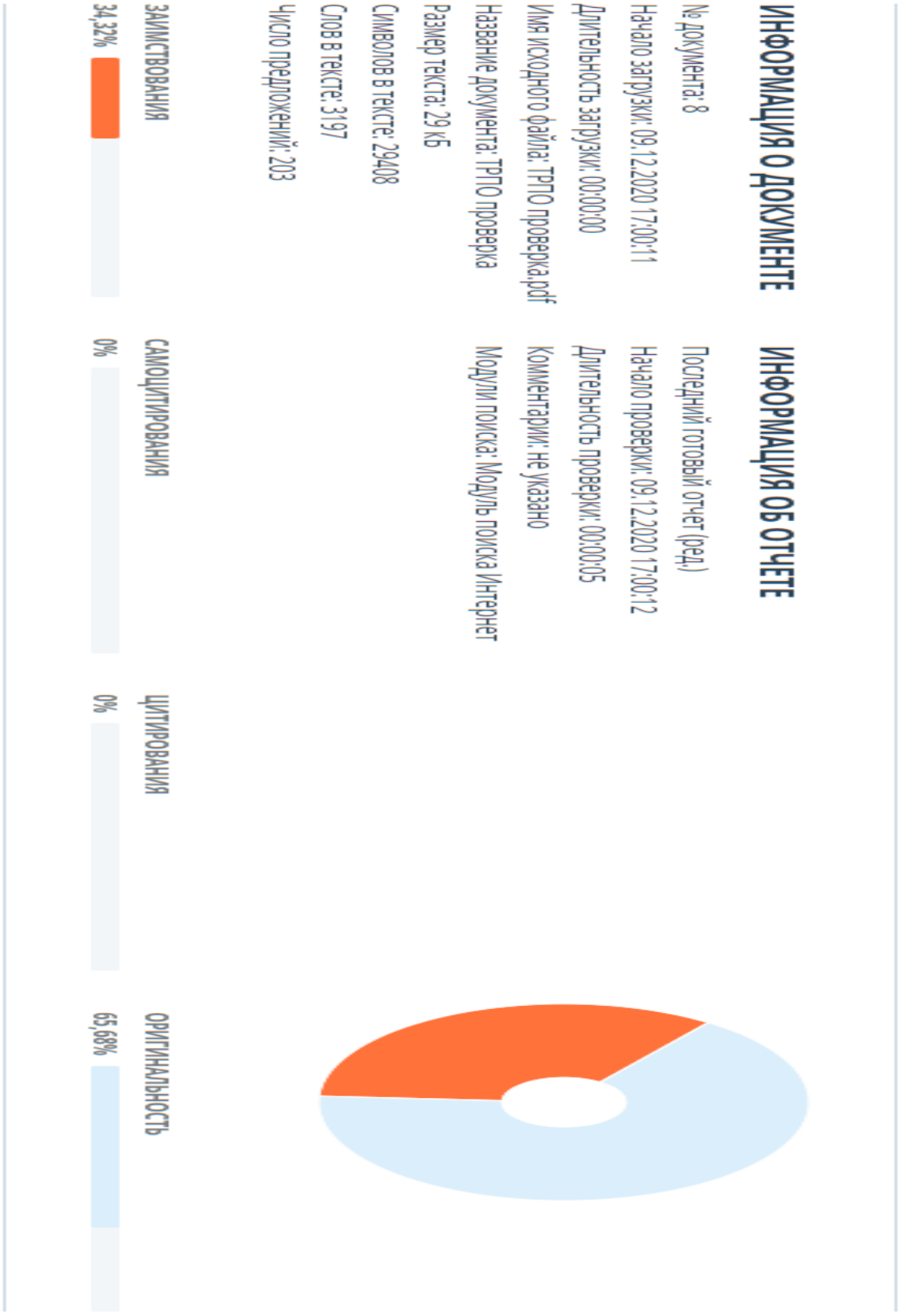
Были получены важные знания и практические навыки как в области использования объектно-ориентированных языков программирования в целом, так и в области построения диаграмм проектирования, отображающих поведение различных организационных структур.

Список использованных источников

1. Ларман, Крэг. Применение UML 2.0 и шаблонов проектирования. Введение в объектно-ориентированный анализ, проектирование и итеративную разработку / КрэгЛарман. - Москва: Гостехиздат, 2017. - 736 с.
2. Роберт А. Максимчук. UML для простых смертных / Роберт А. Максимчук, Эрик Дж. Нейбург. - Москва: СИНТЕГ, 2014. - 272 с.
3. Йордон, Эдвард. Объектно-ориентированный анализ и проектирование систем / Эдвард Йордон , Карл Аргила. - М.: ЛОРИ, 2014. - 264 с.
4. SoloLearn – C# Tutorial. [Электронный ресурс]: - Режим доступа: https://www.sololearn.com/Course/CSharp/ (Дата обращения 13.03.2020).
5. Википедия. [Электронный ресурс]: - Режим доступа: https://ru.wikipedia.org (Дата обращения 17.09.2019).
6. GitHub – NazarKornienko/kursach. [Электронный ресурс]: - Режим доступа: https://github.com/NazarKornienko/kursach (Дата обращения 06.05.2020).
7. Comindware – Нотация BPMN 2.0 [Электронный ресурс]: - Режим доступа: https://comindware.com/ru/blog-нотация-bpmn-2-0-элементы-и-описание/ (Дата обращения 28.02.2020)
8. SysAna– Требования к системе: классификация FURPS+ [Электронный ресурс]: - Режим доступа: https://sysana.wordpress.com/2010/09/16/furps/ (Дата обращения 03.03.2020)

Приложение А

Проверка на антиплагиат



Приложение В

Листинг программы

Листинг cardcheck.cs

```
using System;
using System.IO;
using System.Windows.Forms;

namespace WindowsFormsApp3
{
public partial class cardcheck : Form
{
public cardcheck()
{
InitializeComponent();
DataUsing start = new DataUsing();
cardpincode.Hide();
label2.Hide();
button2.Hide();
}

private void Button1_Click(object sender, EventArgs e)
{
Application.Exit();

}

private void Cardcheck_Load(object sender, EventArgs e)
{

}
class Cardcheck
{
public int trycount;
public bool cardisbelong ;
public bool cardisbloced ;
public Cardcheck(int trycount)
{
this.trycount = trycount;
}
}
Cardcheck t = new Cardcheck(3);
private void Button2_Click(object sender, EventArgs e)
{
Card[] cards = DataBank.cards;
int cardcount =cards.Length;
if (t.cardisbelong == true && t.cardisbloced == false)
{
```

```
for (int i = 0; i < cardcount; i++)
{
if (cardnumber.Text == cards[i].cardnumber)
{
DataBank.cardnumber = i;
if (cardpincode.Text != "")
{
if (cardpincode.Text != Convert.ToString(cards[i].cardpincode))
{
t.trycount -= 1;
MessageBox.Show("введен неправильный пинкод!");
cardpincode.Clear();
}
if (cardpincode.Text == Convert.ToString(cards[i].cardpincode))
{
DataBank.clientid = i;
this.Hide();
clientmenu clientmenu = new clientmenu();
clientmenu.Show();
}
if (t.trycount == 0)
{
MessageBox.Show("карта заблокирована");
string[] block = { cards[i].cardnumber };
File.AppendAllLines("D:\\Загрузки\\cash-machine-master\\cash-
machine-master\\Bank\\Bank\\blockedcards.txt", block);
string[] cardinfo = File.ReadAllLines("D:\\Загрузки\\cash-
machine-master\\cash-machine-master\\Bank\\Bank\\cardinfo.txt");
this.Hide();
Autorization autorization = new Autorization();
autorization.Show();
}
}
}
}
}
}

private void Button3_Click(object sender, EventArgs e)
{
cardpincode.Show();
label2.Show();
button2.Show();
Card[] cards = DataBank.cards;
string[] BlockedCars = File.ReadAllLines("D:\\Загрузки\\cash-
machine-master\\cash-machine-
master\\Bank\\Bank\\blockedcards.txt");
int bcardcount = BlockedCars.Length;
```

```
int cardcount = cards.Length;
t.cardisbelong = false;
t.cardisbloced = false;
for (int i = 0; i < cardcount; i++)
{
if (cardnumber.Text == cards[i].cardnumber)
{
t.cardisbelong = true;
}
}
if (t.cardisbelong == false)
{
MessageBox.Show("извините ваша карта не обслуживается нашим
банком!");
this.Hide();
Autorization autorization = new Autorization();
autorization.Show();
}
for (int i = 0; i < bcardcount; i++)
{
if (cardnumber.Text == BlockedCars[i])
{
t.cardisbloced = true;
MessageBox.Show("извините ваша карта заблокирована!");
this.Hide();
Autorization autorization = new Autorization();
autorization.Show();
}
}
}
}
}
```

Листинг DataBank.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;

namespace WindowsFormsApp3
{
struct Card
{
public string cardnumber;
public string cardpincode;
public int balanse;
```

```
public Card(string cardnumber, string cardpincode, int balanse)
{
this.cardnumber = cardnumber;
this.cardpincode = cardpincode;
this.balanse = balanse;
}
}
static class DataBank
{

public static Card[] cards;
public static int clientcashget;
public static int cardnumber;
public static int clientid;
public static int cardcount;
public static int atmbalance;
public static string atmpincode = "0000";

}
class DataUsing
{
public DataUsing()
{
string[] balance = File.ReadAllLines("D:\\Загрузки\\cash-
machine-master\\cash-machine-
master\\Bank\\Bank\\atmbalance.txt");
DataBank.atmbalance = Convert.ToInt32(balance[0]);
string[] cardinfo = File.ReadAllLines("D:\\Загрузки\\cash-
machine-master\\cash-machine-master\\Bank\\Bank\\cardinfo.txt");
int cardcount = cardinfo.Length / 3;
DataBank.cardcount = cardcount;
Card[] cards = new Card[cardcount];
int b = 0;
for (int i = 0; i < cardinfo.Length-1; i += 3)
{
string cardnumber = cardinfo[i];
string cardpincode = cardinfo[i + 1];
int balanse = int.Parse(cardinfo[i + 2]);
cards[b] = new Card(cardnumber, cardpincode, balanse);
b++;
}
DataBank.cards = cards;

}
}
}
```

Листинг cashget.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;

namespace WindowsFormsApp3
{
public partial class cashget : Form
{
public cashget()
{
InitializeComponent();
}

private void Button7_Click(object sender, EventArgs e)
{
this.Hide();
clientmenu clientmenu = new clientmenu();
clientmenu.Show();
}

private void Button1_Click(object sender, EventArgs e) //10p
{
DataBank.clientcashget = 10;
if (DataBank.cards[DataBank.cardnumber].balanse >=
DataBank.clientcashget)
{
if (DataBank.clientcashget >= DataBank.atmbalance)
{
MessageBox.Show("извините, в банкомате недостаточно средств");
this.Hide();
Autorization autorization = new Autorization();
autorization.Show();
}
else
{
MessageBox.Show("Выньте карту для продолжения");
DataBank.atmbalance -= DataBank.clientcashget;
String[] atmbalance = { Convert.ToString(DataBank.atmbalance) };
string[] cardinfo = File.ReadAllLines("C:\\Bank\\cardinfo.txt");
int balanseID = (DataBank.clientid + 1) * 3 - 1;
cardinfo[balanseID] =
Convert.ToString(DataBank.cards[DataBank.cardnumber].balanse -
DataBank.clientcashget);
```

```
File.WriteAllLines("C:\\Bank\\cardinfo.txt", cardinfo);
File.WriteAllLines("C:\\Bank\\atmbalance.txt", atmbalance);
this.Hide();
cashpay cashpay = new cashpay();
cashpay.Show();
}
}
else
{
MessageBox.Show("на вашем счету недостаточно средств");
this.Hide();
Autorization autorization = new Autorization();
autorization.Show();
}

}

private void Button2_Click(object sender, EventArgs e)
{
DataBank.clientcashget = 50;
if (DataBank.cards[DataBank.cardnumber].balanse >=
DataBank.clientcashget)
{
if (DataBank.clientcashget >= DataBank.atmbalance)
{
MessageBox.Show("извините, в банкомате недостаточно средств");
this.Hide();
Autorization autorization = new Autorization();
autorization.Show();
}
else
{
MessageBox.Show("Выньте карту для продолжения");
DataBank.atmbalance -= DataBank.clientcashget;
String[] atmbalance = { Convert.ToString(DataBank.atmbalance) };
string[] cardinfo = File.ReadAllLines("C:\\Bank\\cardinfo.txt");
int balanseID = (DataBank.clientid + 1) * 3 - 1;
cardinfo[balanseID] =
Convert.ToString(DataBank.cards[DataBank.cardnumber].balanse -
DataBank.clientcashget);
File.WriteAllLines("C:\\Bank\\cardinfo.txt", cardinfo);
File.WriteAllLines("C:\\Bank\\atmbalance.txt", atmbalance);
this.Hide();
cashpay cashpay = new cashpay();
cashpay.Show();
}
}
else
{
```

```
MessageBox.Show("на вашем счету недостаточно средств");
this.Hide();
Autorization autorization = new Autorization();
autorization.Show();
}
}

private void Button3_Click(object sender, EventArgs e)
{
DataBank.clientcashget = 100;
if (DataBank.cards[DataBank.cardnumber].balanse >=
DataBank.clientcashget)
{
if (DataBank.clientcashget >= DataBank.atmbalance)
{
MessageBox.Show("извините, в банкомате недостаточно средств");
this.Hide();
Autorization autorization = new Autorization();
autorization.Show();
}
else
{
MessageBox.Show("Выньте карту для продолжения");
DataBank.atmbalance -= DataBank.clientcashget;
String[] atmbalance = { Convert.ToString(DataBank.atmbalance) };
string[] cardinfo = File.ReadAllLines("D:\\Загрузки\\cash-
machine-master\\cash-machine-master\\Bank\\Bank\\cardinfo.txt");
int balanseID = (DataBank.clientid + 1) * 3 - 1;
cardinfo[balanseID] =
Convert.ToString(DataBank.cards[DataBank.cardnumber].balanse -
DataBank.clientcashget);
File.WriteAllLines("D:\\Загрузки\\cash-machine-master\\cash-
machine-master\\Bank\\Bank\\cardinfo.txt", cardinfo);
File.WriteAllLines("D:\\Загрузки\\cash-machine-master\\cash-
machine-master\\Bank\\Bank\\atmbalance.txt", atmbalance);
this.Hide();
cashpay cashpay = new cashpay();
cashpay.Show();
}
}
else
{
MessageBox.Show("на вашем счету недостаточно средств");
messagebox.show
MessageBox.Show

this.Hide();
Autorization autorization = new Autorization();
autorization.Show();
```

```
}
}

private void Button4_Click(object sender, EventArgs e)
{
DataBank.clientcashget = 200;
if (DataBank.cards[DataBank.cardnumber].balanse >=
DataBank.clientcashget)
{
if (DataBank.clientcashget >= DataBank.atmbalance)
{
MessageBox.Show("извините, в банкомате недостаточно средств");
this.Hide();
Autorization autorization = new Autorization();
autorization.Show();
}
else
{
MessageBox.Show("Выньте карту для продолжения");
DataBank.atmbalance -= DataBank.clientcashget;
String[] atmbalance = { Convert.ToString(DataBank.atmbalance) };
string[] cardinfo = File.ReadAllLines("D:\\Загрузки\\cash-
machine-master\\cash-machine-master\\Bank\\Bank\\cardinfo.txt");
int balanseID = (DataBank.clientid + 1) * 3 - 1;
cardinfo[balanseID] =
Convert.ToString(DataBank.cards[DataBank.cardnumber].balanse -
DataBank.clientcashget);
File.WriteAllLines("D:\\Загрузки\\cash-machine-master\\cash-
machine-master\\Bank\\Bank\\cardinfo.txt", cardinfo);
File.WriteAllLines("D:\\Загрузки\\cash-machine-master\\cash-
machine-master\\Bank\\Bank\\atmbalance.txt", atmbalance);
this.Hide();
cashpay cashpay = new cashpay();
cashpay.Show();
}
}
else
{
MessageBox.Show("на вашем счету недостаточно средств");
this.Hide();
Autorization autorization = new Autorization();
autorization.Show();
}
}

private void Button5_Click(object sender, EventArgs e)
{
DataBank.clientcashget = 500;
```

```
if (DataBank.cards[DataBank.cardnumber].balanse >=
DataBank.clientcashget)
{
if (DataBank.clientcashget >= DataBank.atmbalance)
{
MessageBox.Show("извините, в банкомате недостаточно средств");
this.Hide();
Autorization autorization = new Autorization();
autorization.Show();
}
else
{
MessageBox.Show("Выньте карту для продолжения");
DataBank.atmbalance -= DataBank.clientcashget;
String[] atmbalance = { Convert.ToString(DataBank.atmbalance) };
string[] cardinfo = File.ReadAllLines("C:\\Bank\\cardinfo.txt");
int balanseID = (DataBank.clientid + 1) * 3 - 1;
cardinfo[balanseID] =
Convert.ToString(DataBank.cards[DataBank.cardnumber].balanse -
DataBank.clientcashget);
File.WriteAllLines("D:\\Загрузки\\cash-machine-master\\cash-
machine-master\\Bank\\Bank\\cardinfo.txt", cardinfo);
File.WriteAllLines("D:\\Загрузки\\cash-machine-master\\cash-
machine-master\\Bank\\Bank\\atmbalance.txt", atmbalance);
this.Hide();
cashpay cashpay = new cashpay();
cashpay.Show();
}
}
else
{
MessageBox.Show("на вашем счету недостаточно средств");
this.Hide();
Autorization autorization = new Autorization();
autorization.Show();
}
}

private void Button6_Click(object sender, EventArgs e)
{
DataBank.clientcashget = 1000;
if (DataBank.cards[DataBank.cardnumber].balanse >=
DataBank.clientcashget)
{
if (DataBank.clientcashget >= DataBank.atmbalance)
{
MessageBox.Show("извините, в банкомате недостаточно средств");
this.Hide();
Autorization autorization = new Autorization();
```

```
autorization.Show();
}
else
{
MessageBox.Show("Выньте карту для продолжения");
DataBank.atmbalance -= DataBank.clientcashget;
String[] atmbalance = { Convert.ToString(DataBank.atmbalance) };
string[] cardinfo = File.ReadAllLines("D:\\Загрузки\\cash-
machine-master\\cash-machine-master\\Bank\\Bank\\cardinfo.txt");
int balanseID =(DataBank.clientid+1)*3-1;
cardinfo[balanseID] =
Convert.ToString(DataBank.cards[DataBank.cardnumber].balanse -
DataBank.clientcashget);
MessageBox.Show(" "+ cardinfo[balanseID] + "\n " +
Convert.ToString(balanseID) );
File.WriteAllLines("D:\\Загрузки\\cash-machine-master\\cash-
machine-master\\Bank\\Bank\\cardinfo.txt", cardinfo);
File.WriteAllLines("C:\\Bank\\atmbalance.txt", atmbalance);
this.Hide();
cashpay cashpay = new cashpay();
cashpay.Show();
}
}
else
{
MessageBox.Show("на вашем счету недостаточно средств");
this.Hide();
Autorization autorization = new Autorization();
autorization.Show();
}
}

private void Cashget_Load(object

sender, EventArgs e)
{

}
}
}
```