Реферат

Курсовой проект: 49 страниц, 29 рисунков, 1 таблица, 7 используемых источников, 2 приложения.

ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ, ПРОЕКТИРОВАНИЕ, ХОЛОДИЛЬНИК, ПОСЛЕДОВАТЕЛЬНОСТИ, МОДЕЛЬ, КЛАСС, UML, ДИАГРАММА ГАНТА, СЕТЕВАЯ ДИАГРАММА, ДИАГРАММЫ IDEF0, IDEF1X, IDEF3.

В процессе выполнения данного задания было разработано программное обеспечение эмулирующие процесс работы холодильника, который стоит в доме каждого из нас.

Целью работы является разработка проекта эмулирующего работу холодильника, с использованием диаграмм разного вида, в полной мере описывающих как внутреннее устройство исследуемой системы, так и всевозможные взаимодействия между её компонентами.

В конечном итоге были получены диаграммы, обладающие исчерпывающей информацией о программном обеспечение эмулирующего работу холодильника, регулирующего доступ в помещение. К ним относятся: диаграмма Ганта, UML-диаграммы, IDEF0-диаграмма, IDEF3 диаграмма. Проведено тестирование.

Содержание

Введение

В современном мире разработка новых технологий не стоит на месте, для проверки задуманных разработок, создают прототипы, благодаря которым можно испытать запланированный функционал. Прототипы бывают разных типов, физические или цифровые, все зависит от характеристик проекта и возможности реализации тех или иных целей.

В своем курсовом проекте я разрабатываю цифровой прототип, программное обеспечение эмулирующее работа холодильника, благодаря которому мы можем проверить на практике работоспособность заданных функций.

Данных курсовой проект написан с нуля, так как эта тема ещё никем не рассматривалась. Мной была спроектирована программа, для нее был разработан функционал и интерфейс. Были построены несколько диаграм IDEF0, диаграмма IDEF3, также диаграмма UML, была проведена оценка трудозатрат, совершена планировка работ и как следствие диаграмма Ганта и сетевая диаграмма.

1 Формулировка задачи

Задачей данного курсового проекта является разработка программы для холодильника.

Холодильник должен состоять из следующих компонентов:

– передней стороны с ручкой и дисплея управления;

– дисплея, с помощью которого можно управлять температурным режимом и узнавать актуальные температуры;

– внутреннего пространства холодильника, в котором можно управлять продуктами.

Указанное программное обеспечение должно предоставлять оператору следующий набор функций управления:

– холодильник;

– возможность мониторинга состояния;

– возможность управлять температурой;

– возможность управлять продуктами и вести их учет.

Любой пользователь холодильника имеет возможность взаимодействовать с ним, он может: изменить температуру, выставить заранее заданный режим, открывать дверь и взаимодействовать с продуктами. Любое действие происходит последовательно, для взаимодействия с продуктами, необходимо открыть дверь, для изменения режимов, закрыть дверь. Каждое взаимодействие с температурным режимом будет сопровождаться текстовым сообщением, благодаря которому мы можем узнать о статусе выполнения необходимой функции.

2 Управление проектом

2.1 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. Выглядит она 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.

Диаграмма Ганта для проекта «Программное обеспечение, эмулирующие работу холодильника» приведена на рисунках 1 и 2.

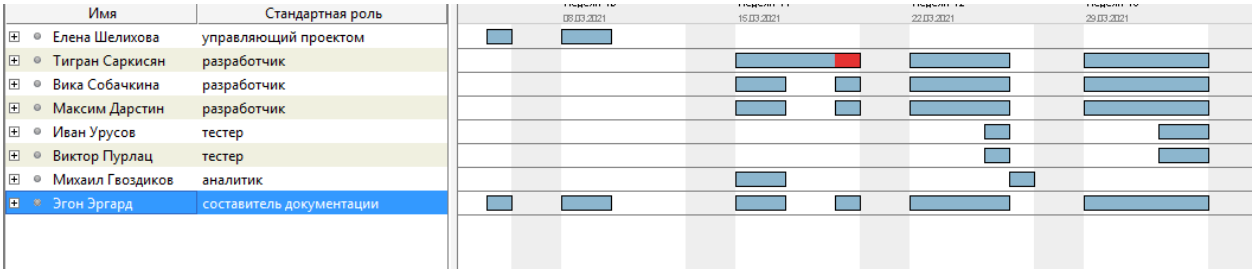


Рисунок 1 – Распределение работников по этапам проекта

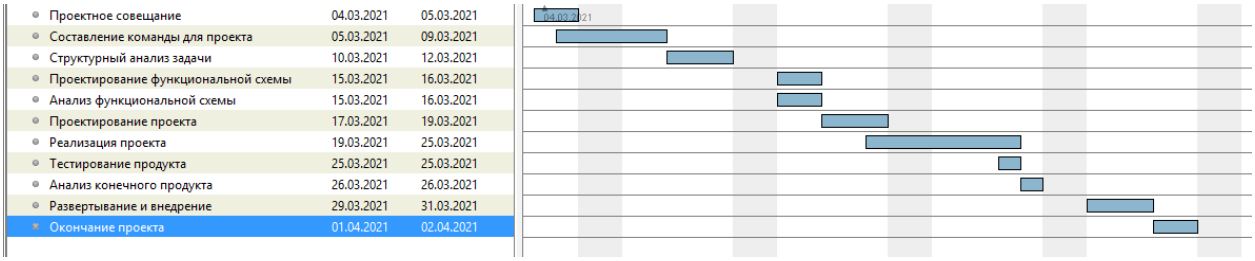


Рисунок 2 – Диаграмма Ганта

2.2 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. Сетевая диаграмма проекта «Холодильник» показана на рисунке 3.

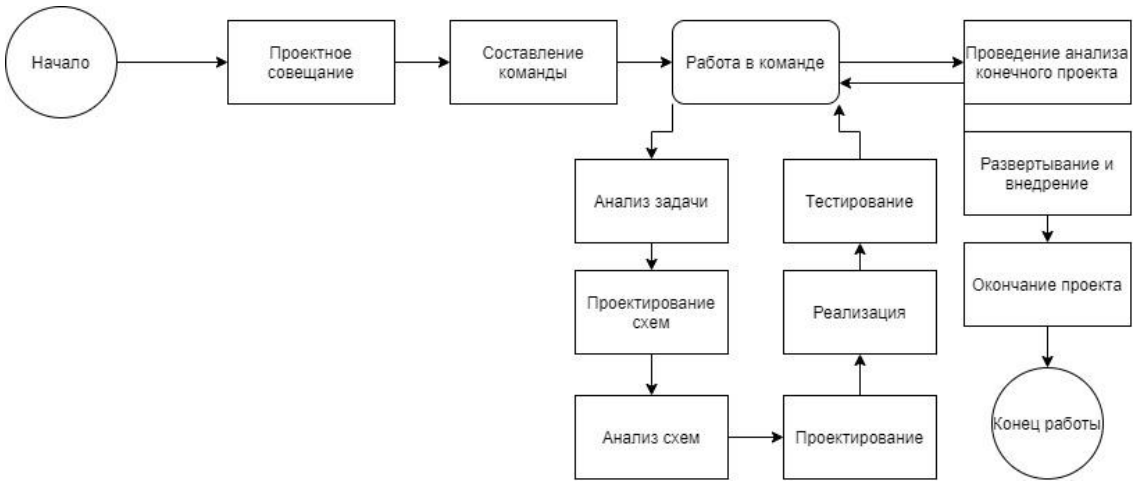


Рисунок 3 –Сетевая диаграмма

3 Создание модели As-Is в методологии IDEF0

Для того чтобы хорошо оценить возможности, разрабатываемого ПО нужно 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 уязвимые места, какие будут плюсы и минусы, протекающих в ней бизнес-процессов относительно конкурентов. Использование данной модели поможет 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 – методология функционального моделирования (англ. Function modeling) и графическая нотация, предназначенная для формализации и описания бизнес-процессов. Отличительной особенностью IDEF0 является её акцент на соподчинённость объектов. В IDEF0 рассматриваются логические отношения между работами, а не их временная последовательность (поток работ).

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 системы «Холодильник».

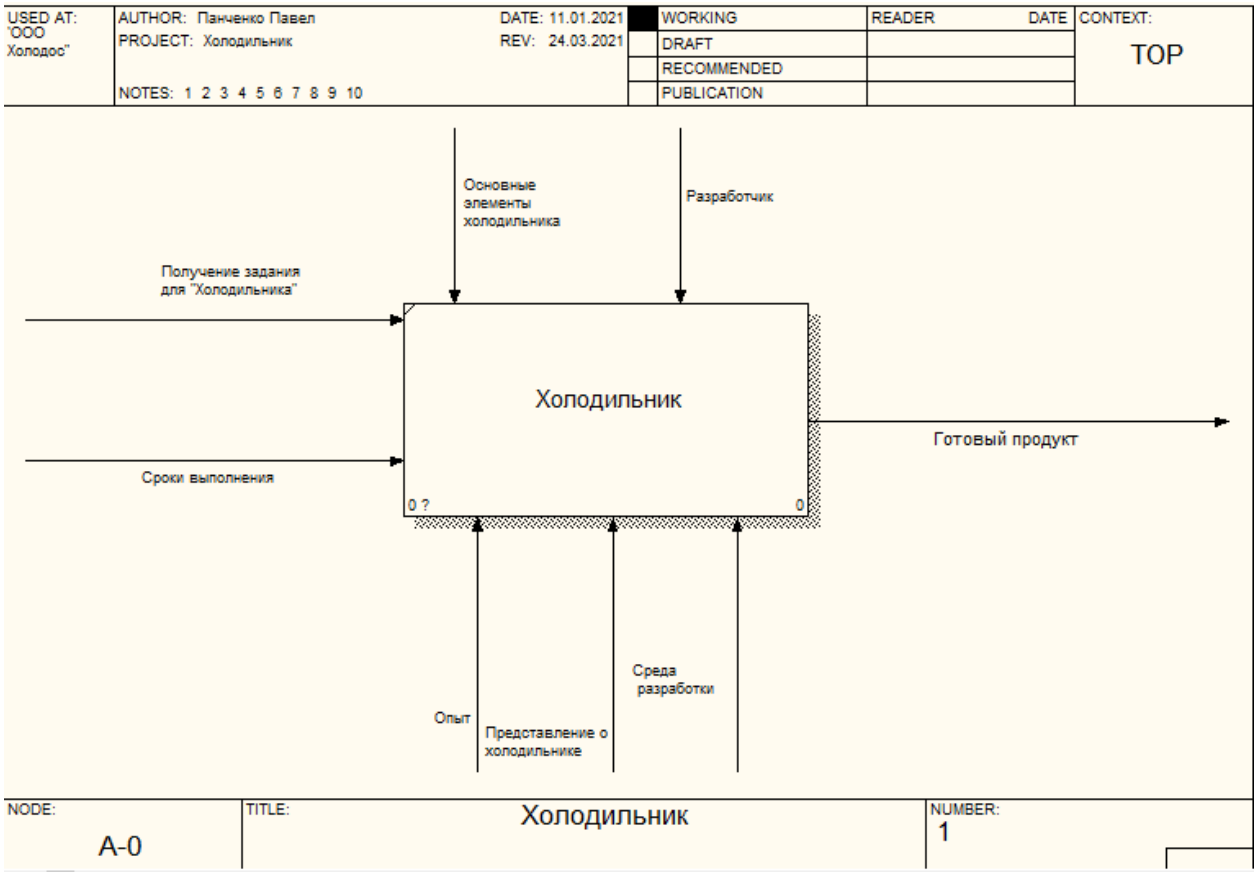


Рисунок 4 – Контекстная диаграмма IDEF0 окончательного варианта проектирования

На диаграмме представлен процесс проектирования задания, описано получение задания, варианты его проектирования и необходимых для этого средств.

Управление происходит благодаря нормативной документации и методических указаниям.

Механизмом реализации работы системы являются аппаратное и программное обеспечение холодильника.

Результатом деятельности системы является визуальное сообщение об окончании действия, а также визуальные изменения на цифровой панели передней дверцы холодильника .

На рисунке 5 с помощью диаграммы IDEF0 я показал процесс разработки и реализации программы с задуманным функционалом.

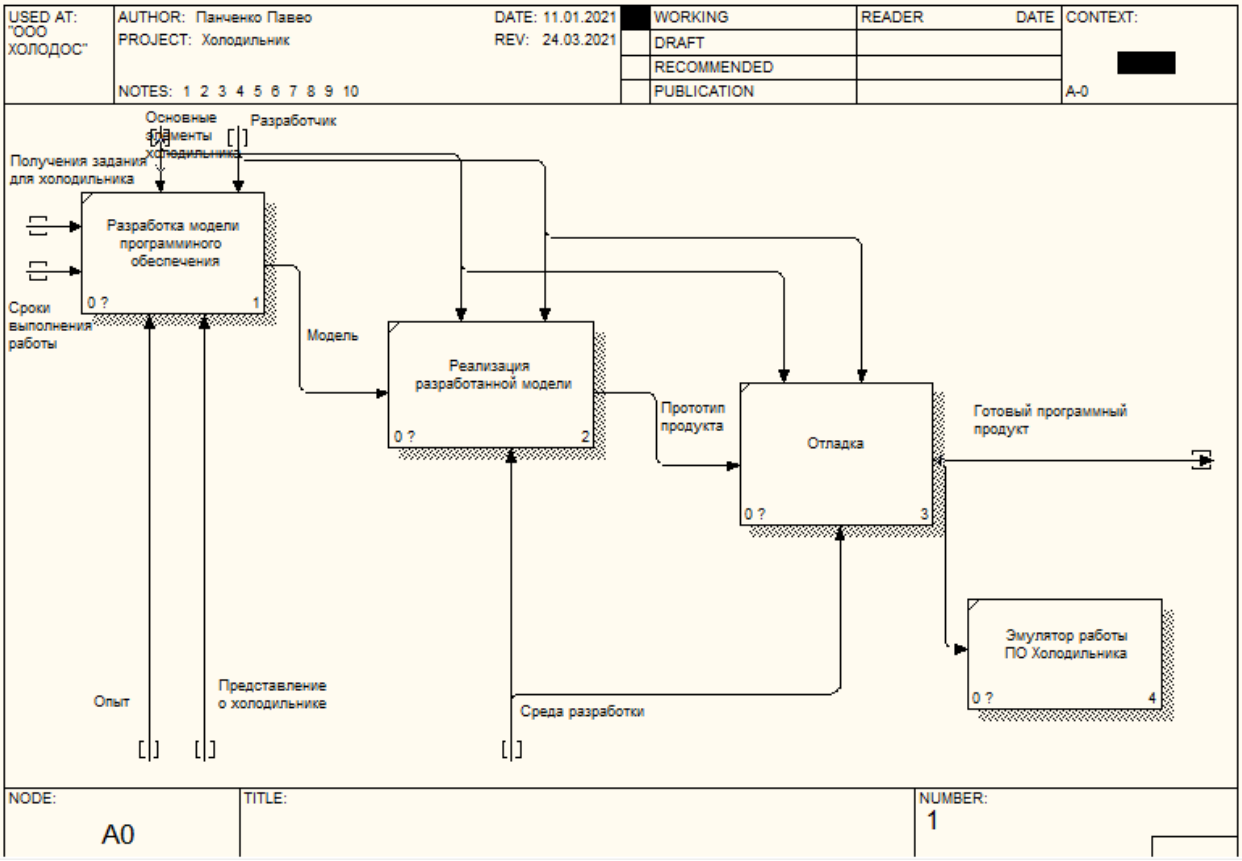


Рисунок 5 – Диаграмма программирования и реализации проекта

Полученная модель системы может быть представлена в более подробном виде путём разбиения на большее количество составных элементов т. е. применения декомпозиции.

На рисунке 6 изображена модель «Холодильник» после декомпозиции.

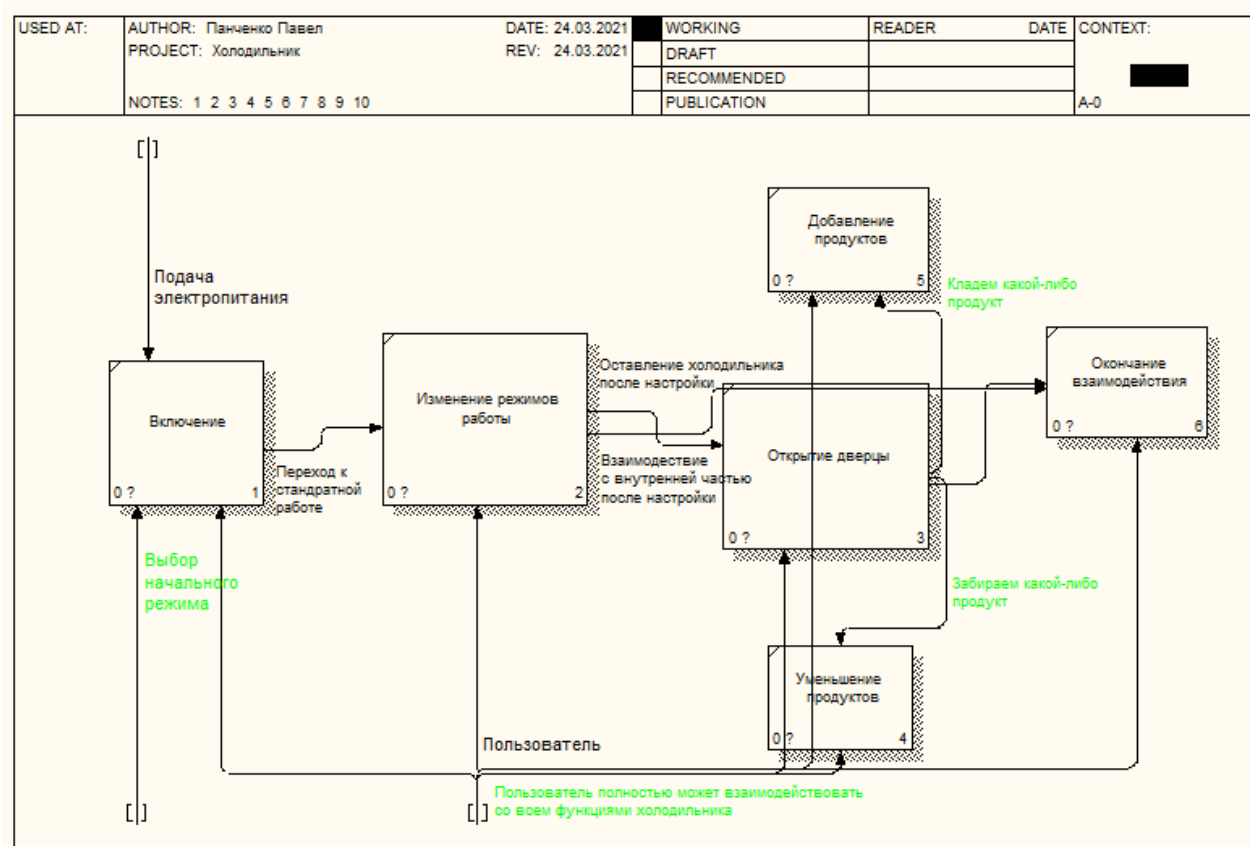


Рисунок 6 – Декомпозиция IDEF0 1-го уровня для системы автоматизации холодильника

4 Метод описания процессов 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 водном процессе. Техника описания набора данных IDEF3 является частью структурного анализа. В отличие от некоторых методик описаний процессов IDEF3 не ограничивает аналитика чрезмерно жесткими рамками синтаксиса, что может привести к созданию неполных или противоречивых моделей.IDEF3 дополняет IDEF0 и содержит все необходимое для построения моделей, которые в дальнейшем могут быть использованы для имитационного анализа.

Основные описательные блоки диаграммы IDEF3:

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

– Связи показывают взаимоотношение 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. Окончание одной работы может служить сигналом к началу нескольких работ, или же одна работа для своего запуска может ожидать окончание нескольких работ.

Типы перекрестков описаны на рисунке 7.

Обозначение	Наименование	Смысл в случае слияния стрелок	Смысл в случае разветвления стрелок
&	Asynchronous AND	Все предшествующие процессы должны быть завершены	Все следующие процессы должны быть запущены
&	Synchronous AND	Все предшествующие процессы завершены одновременно	Все следующие процессы запускаются одновременно
O	Asynchronous OR	Один или несколько предшествующих процессов должны быть завершены	Один или несколько следующих процессов должны быть запущены
O	Synchronous OR	Один или несколько предшествующих процессов завершены одновременно	Один или несколько следующих процессов запускаются одновременно
X	XOR (Exclusive OR)	Только один предшествующий процесс завершен	Только один следующий процесс запускается

Рисунок 7 – Типы перекрестков

Для системы эмулирующей процесс работы холодильника диаграмма IDEF3 отображена на рисунке 8.

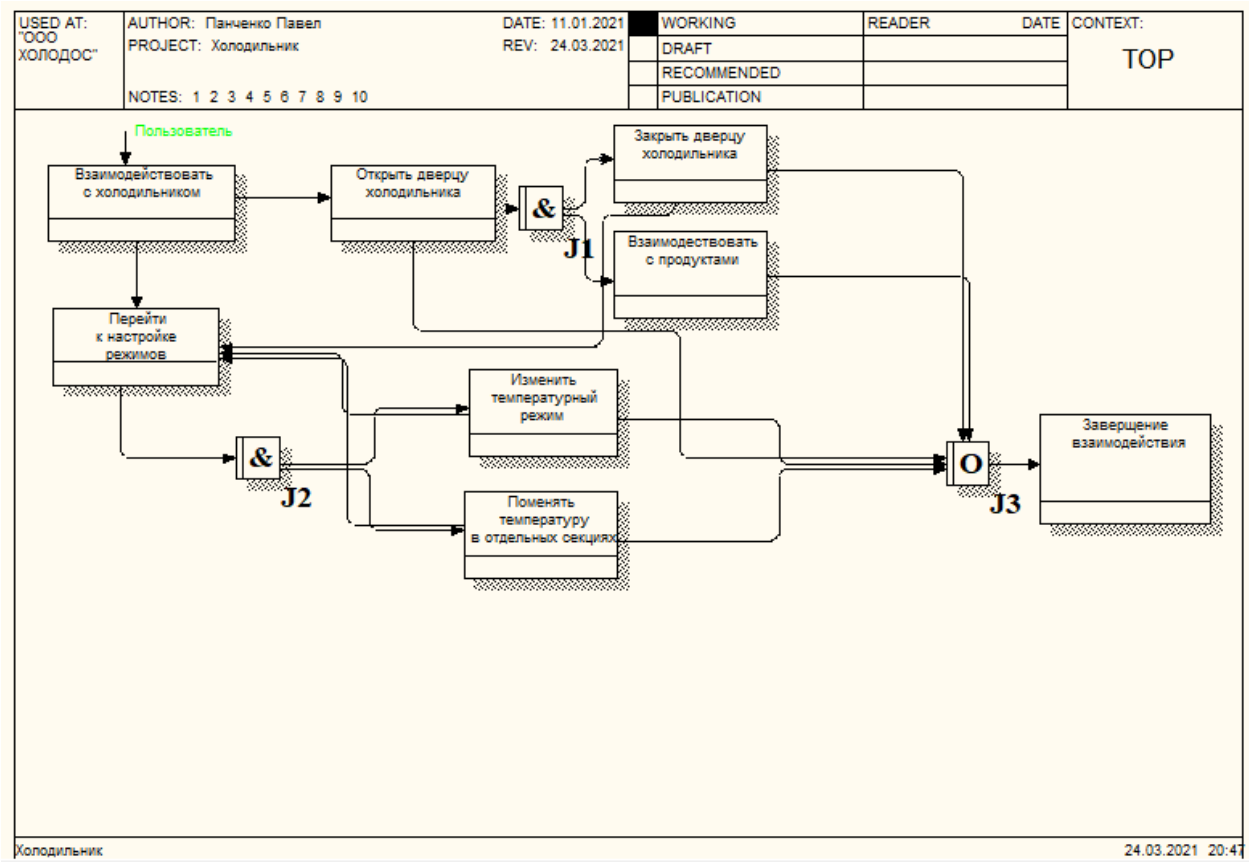


Рисунок 8 – Диаграмма IDEF3 – описание алгоритма взамодествия с системой “Холодильник”

5 Методология IDEF1X

IDEF1X является методом для разработки реляционных баз данных и использует условный синтаксис, специально разработанный для удобного построения концептуальной схемы.

IDEF1X позволяет строить семантические модели данных, которые могут служить для поддержки управления данными как ресурсом, интеграции информационных систем и построения компьютерных баз данных. Этот стандарт является частью семейства языков моделирования IDEF в области программной инженерии.

Рисунок 10 содержит диаграмму в методологии IDEF1X для проекта «Холодильник».

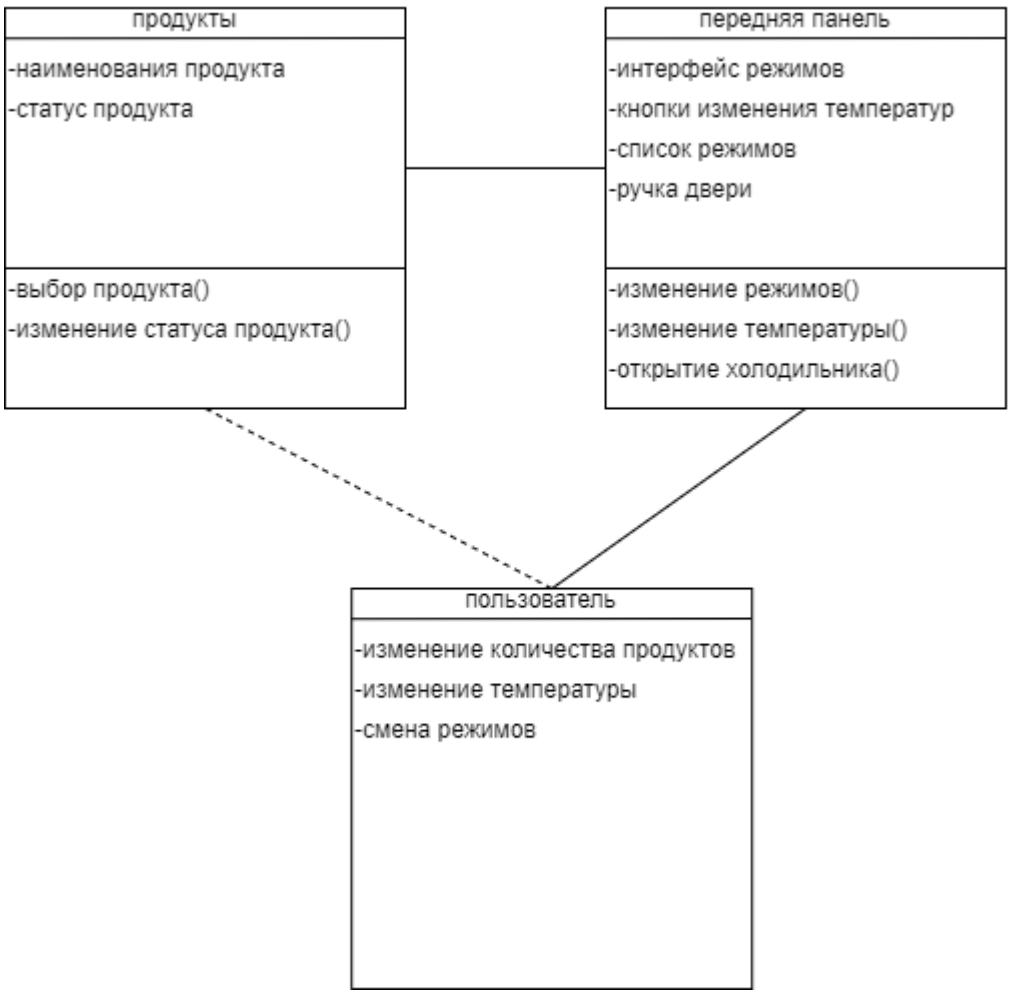


Рисунок 9 – Диаграмма IDEF1X

6 Оценка трудоемкости методом функциональных точек

Оценка и анализ трудозатрат очень актуально и востребовано, в настоящее время. Ведь прежде разрабатывать ПО очень важно знать стоимость, дабы суметь сократить и оптимизировать затраты, не уменьшив качество продукта.

Имеются разные методы решения задачи оценки трудозатрат, в своей работе я исследовал метод функциональных точек.

Метод предназначен для оценки на основе логической модели объема программного продукта количеством функционала, востребованного заказчиком и поставляемого разработчиком. Несомненным достоинством метода является то, что измерения не зависят от технологической платформы, на которой будет разрабатываться продукт, и он обеспечивает единообразный подход к оценке всех проектов в компании.

Для получения функционально-ориентированных метрик (FP-метрик) используются функциональные и концептуальные модели будущей системы (модели IDEF0 (рисунки 5-7) и IDEF1X (рисунок 9)). Основывается процесс получения функционально-ориентированных метрик на функциональной модели системы (модели бизнес-процессов). Рассматриваются только наиболее значимые процессы, соответствующие основным функциям разрабатываемого программного продукта (например, перечисленным в техническом задании). Каждый бизнес-процесс имеет входные и выходные данные, находящие свое отражение в концептуальной модели системы (например, соответствующие сущностям в моделях «сущность-связь»).

На рисунке 10 вы можете увидеть последовательность шагов процедурного анализа по этому способу.

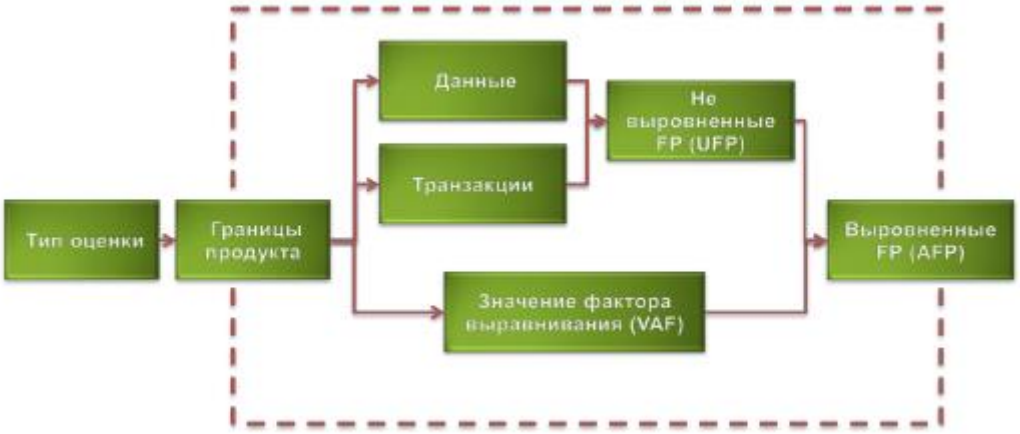


Рисунок 10 – Анализ по методу функциональных точек

Наименование функции	Количество функциональных точек, соответствующее категории функции	Количество функциональных точек
1. Определение количества выводов	6*4	24
2. Определение количества вводов	6*4	24
3. Определение количестваопросов вывода	4*4	16
4. Определение количества опросов ввода	1*4	4
5. Определение количества файлов	6*7	42
6. Определение количества интерфейсов	7*7	49
Общее количество функциональных точек		159

Таблица 1 – Расчет функциональных точек

Уровень влияния факторов внешней среды:

W=0.65+(0.01*30) = 0,95

Уточненное количество функциональных точек:

R(F)=159*0.95=151,05

Размерность ПО для C#:

R(Loc)=151,05*5,4=815,67

где Loc – среднее количество операторов конкретного языка программирования, требующегося для реализации функциональной точки.

Расчёт заработной платы для каждого работника, занятого в проекте отображен на рисунке 11.

Расчёт зарплаты участникам проекта :

Елена Шелихова	45 000 рублей
Тигран Саркисян	33 000 рублей
Максим Дарстин	33 000 рублей
Иван Урусов	33 000 рублей
Виктор Пурлац	26 000 рублей
Михаил Гвоздиков	40 000 рублей
Эгон Эргард	15 000 рублей

Рисунок 11 – Оценка заработной платы работников

7 UML-диаграммы

UML – унифицированный язык моделирования (Unified Modeling Language) – это система обозначений, которую можно применять для объектно-ориентированного анализа и проектирования. Его можно использовать для визуализации, спецификации, конструирования и документирования программных систем.

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.

Создание UML началось в октябре 1994 г., когда Джим Рамбо и Гради Буч из Rational Software Corporation стали работать над объединением своих методов OMT и Booch. В настоящее время консорциум пользователей UML Partners включает в себя представителей таких грандов информационных технологий, как Rational Software, Microsoft, IBM, Hewlett-Packard, Oracle, DEC, Unisys, IntelliCorp, Platinum Technology.

7.1 Диаграмма вариантов использования

Понятие варианта использования (usecase) впервые ввел Ивар Якобсон и придал ему такую значимость, что в настоящее время вариант использования превратился в основной элемент разработки и планирования проекта.

Вариант использования представляет собой последовательность действий (транзакций), выполняемых системой в ответ на событие, инициируемое некоторым внешним объектом (действующим лицом). Вариант использования описывает типичное взаимодействие между пользователем и системой. В простейшем случае вариант использования

определяется в процессе обсуждения с пользователем тех функций, которые он хотел бы реализовать.

Действующее лицо (actor) – это роль, которую пользователь играет по отношению к системе. Действующие лица представляют собой роли, а не конкретных людей или наименования работ. Несмотря на то, что на диаграммах вариантов использования они изображаются в виде стилизованных человеческих фигурок, действующее лицо может также быть внешней системой, которой необходима некоторая информация от данной системы.

Поддерживается несколько типов связей между вариантами использования и действующими лицами:

– связь коммуникации – это связь между вариантом использования и действующим лицом;
– связь включения применяется в тех ситуациях, когда имеется какой-либо фрагмент поведения системы, который повторяется более чем в одном варианте использования;
– связь расширения применяется при описании изменений в нормальном поведении системы;
– с помощью связи обобщения показывают, что у нескольких действующих лиц имеются общие черты.

Для проекта «Холодильник» диаграмма вариантов использования представлена на рисунке 12.

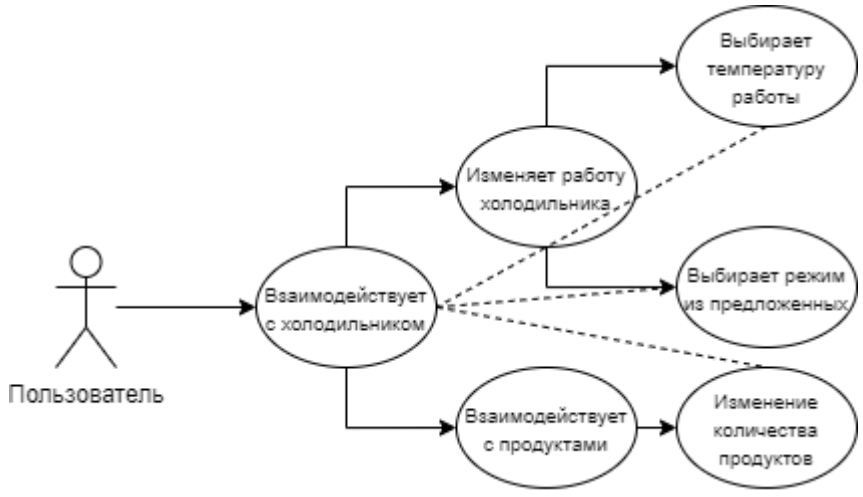


Рисунок 12 – Диаграмма вариантов использования

7.2 Диаграмма последовательности

Диаграмма последовательности (рисунок 13) отражает поток событий, происходящих в рамках варианта использования.

Все действующие лица показаны в верхней части диаграммы. Стрелки соответствуют сообщениям, передаваемым между действующим лицом и объектом или между объектами для выполнения требуемых функций.

На диаграмме последовательности объект изображается в виде прямоугольника, от которого вниз проведена пунктирная вертикальная линия. Эта линия называется линией жизни (lifeline) объекта. Она представляет собой фрагмент жизненного цикла объекта в процессе взаимодействия.

Каждое сообщение представляется в виде стрелки между линиями жизни двух объектов. Сообщения появляются в том порядке, как они показаны на странице сверху вниз. Каждое сообщение помечается как минимум именем сообщения. При желании можно добавить также аргументы и некоторую управляющую информацию.

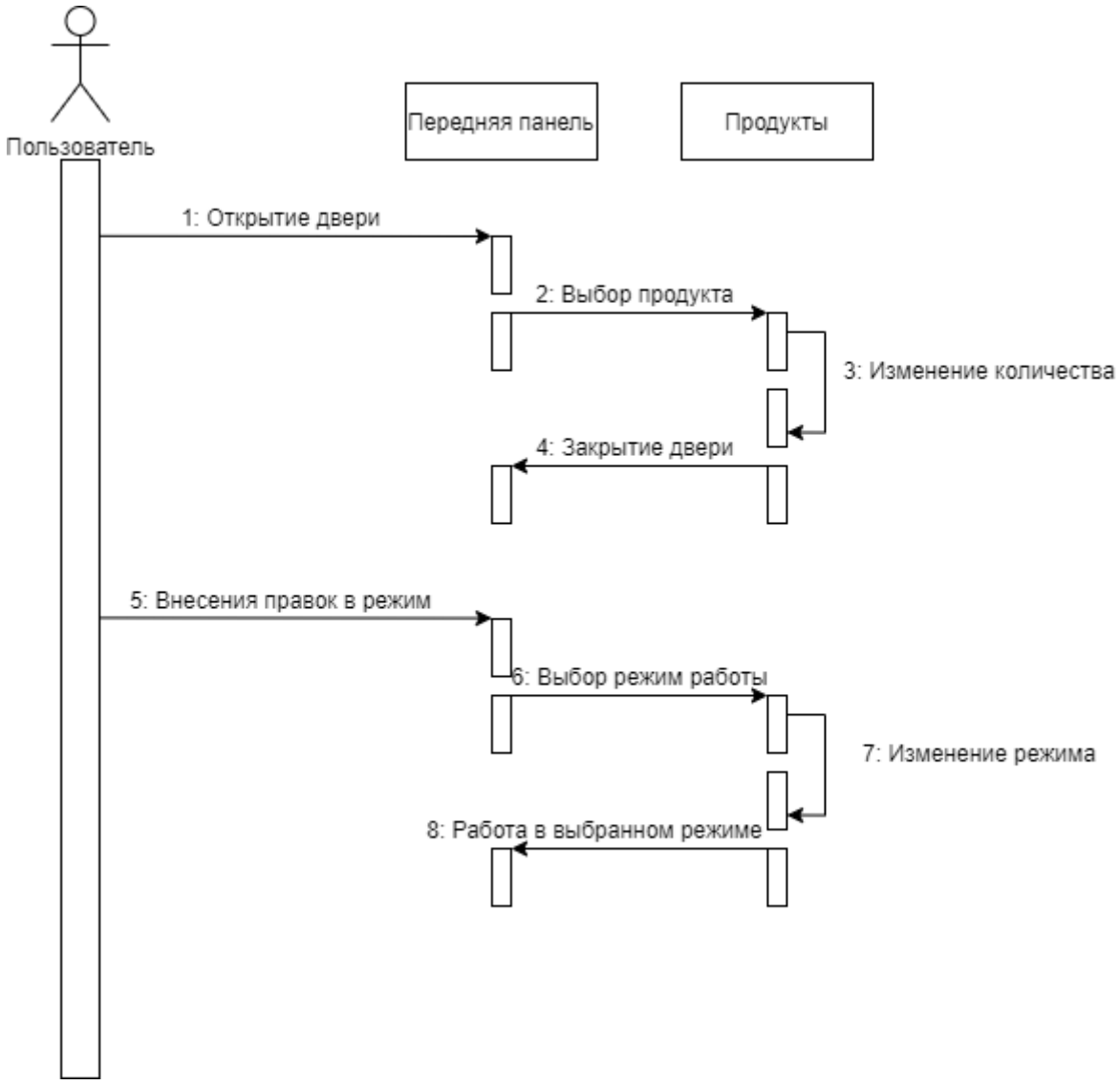


Рисунок 13 – Диаграмма последовательности

7.3 Диаграмма кооперации

Диаграммы кооперации (рисунок 14) отображают поток событий через конкретный сценарий варианта использования, упорядочены по времени, а кооперативные диаграммы больше внимания заостряют на связях между объектами.

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

Для примера я возьму сценарий по изменению температурного режима на разморозку и полную очистку холодильника от продуктов.

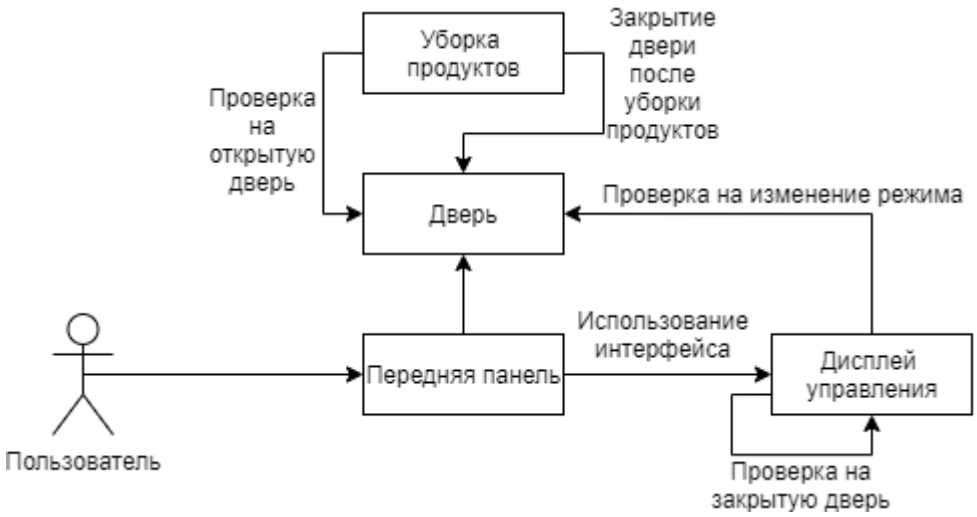


Рисунок 14 – Диаграмма кооперации

7.4 Диаграмма классов

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

Диаграмма классов UML - это граф, узлами которого являются элементы статической структуры проекта (классы, интерфейсы), а дугами - отношения между узлами (ассоциации, наследование, зависимости).

В диаграммах классов я использовал классы представленные на картинке 15.

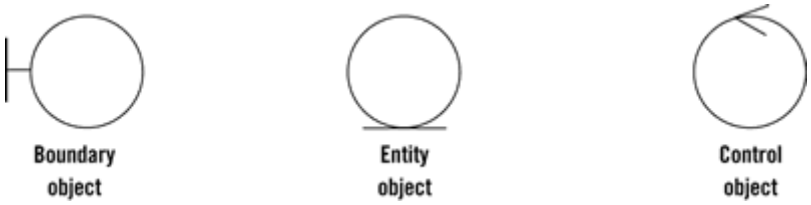


Рисунок 15 – Типы диаграмм классов

Диаграмма классов для проекта «Холодильник» приведена на рисунке 16.

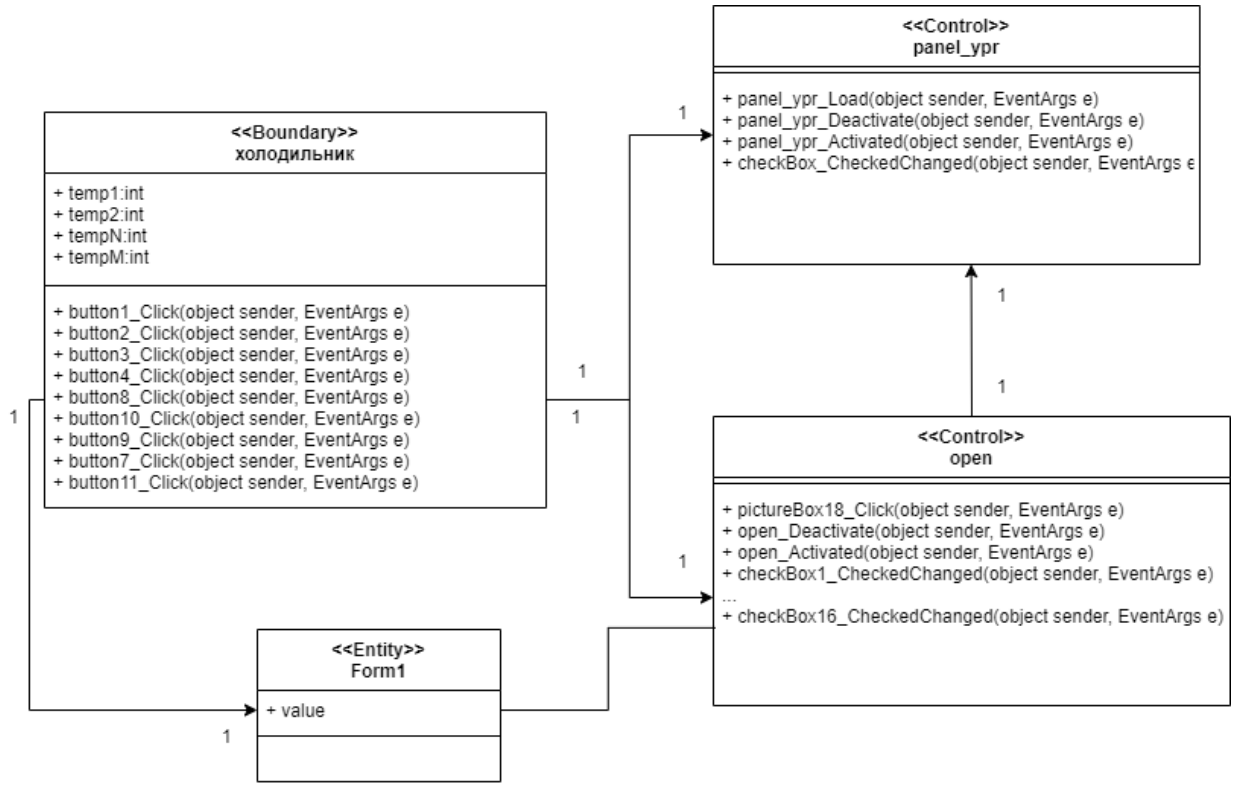


Рисунок 16 – Диаграмма классов

7.5 Диаграмма состояний

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

Существует много форм диаграмм состояний, незначительно отличающихся друг от друга семантикой.

На диаграмме имеются два специальных состояния – начальное и конечное. Начальное состояние выделено черной точкой, оно соответствует состоянию объекта, когда он только что был создан. Конечное состояние обозначается черной точкой в белом кружке, оно соответствует состоянию объекта непосредственно перед его уничтожением. На диаграмме состояний

может быть одно и только одно начальное состояние. В то же время, может быть столько конечных состояний, сколько вам нужно, или их может не быть вообще. Когда объект находится в каком-то конкретном состоянии, могут выполняться различные процессы. Процессы, происходящие, когда объект находится в определенном состоянии, называются действиями.

С состоянием можно связывать данные пяти типов: деятельность, входное действие, выходное действие, событие и история состояния.

Рисунок 17 содержит диаграмму состояний для проекта «Холодильник».

Рисунок 17 – Диаграмма состояний

7.6 Объединённая диаграмма компонентов и размещения

Диаграмма размещения (deployment diagram) 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 быть и мэйнфреймом.

Диаграмма размещения показывает физическое расположение сети и местонахождение в ней различных 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 и 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.

В некоторых случаях допускается размещать диаграмму компонентов на диаграмме развертывания. Это позволяет показать какие компоненты выполняются и на каких узлах. Такая диаграмма для настоящего проекта показана на рисунке 18.

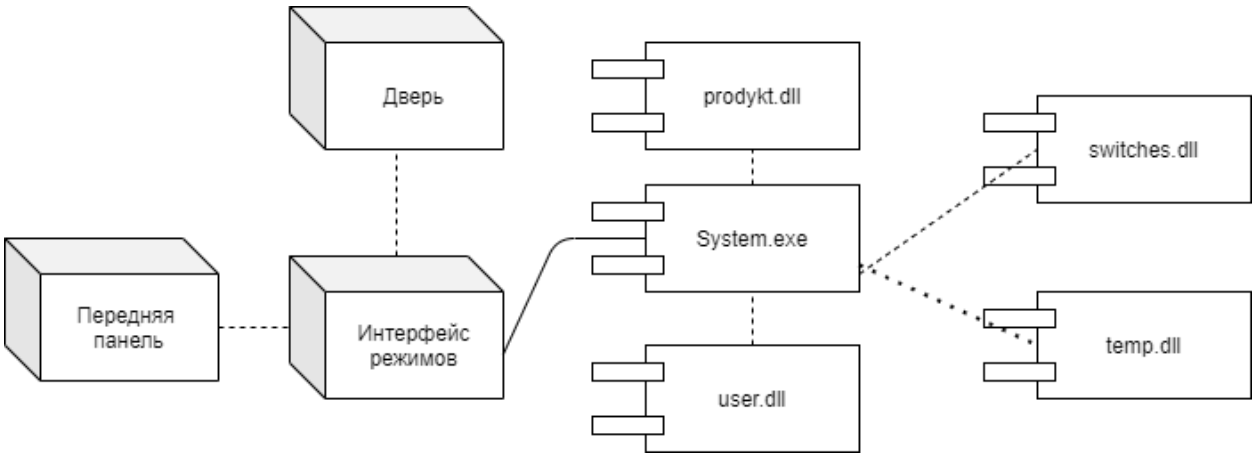


Рисунок 18 – Объединённая диаграмма компонентов и размещения

8 Результаты машинного тестирования программы

При запуске программы открывается главное окно, на котором показаны температуры холодильника, а также наличие продуктов определенной категории (рисунок 19).

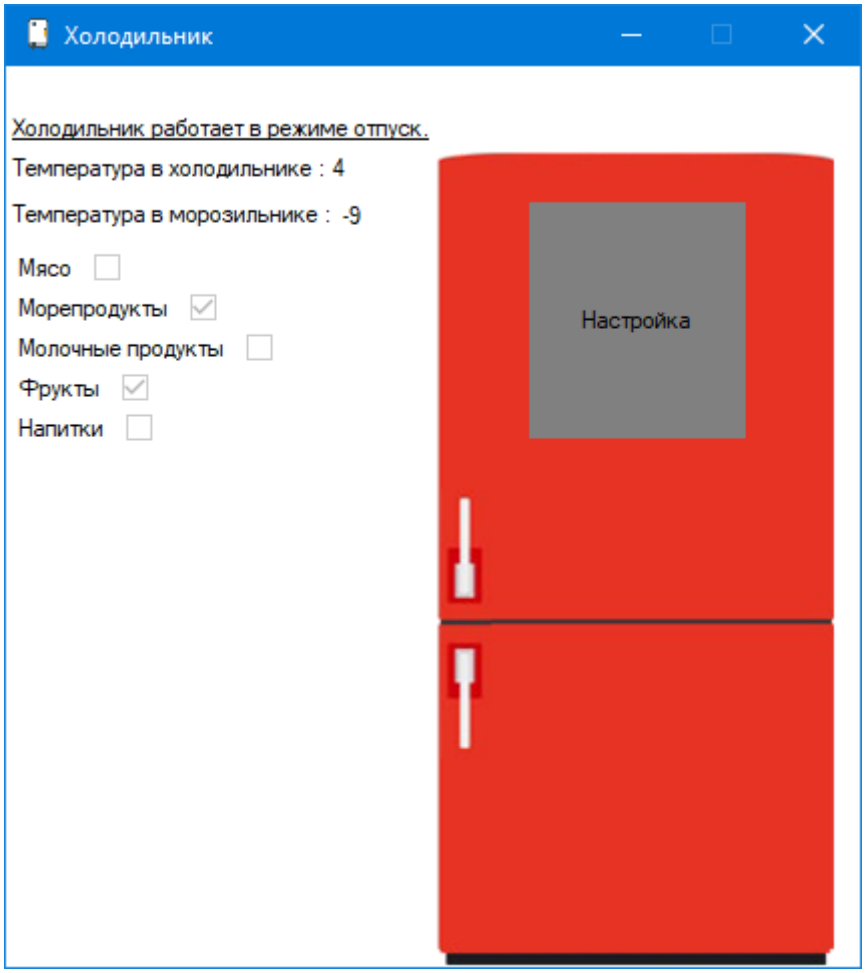


Рисунок 19 – Стартовый экран приложения

При нажатии на окно “Настройки”, появляется панель изменения температуры и режимов. При нажатие стрелочек под каждым показателем температура меняется в определенном отделе (рисунок 20).

Рисунок 20 – Окно панели управления холодильником

При выборе любого режима, появится сообщение об успешной смене режима или может появится ошибка, при несоблюдении условий эксплуатации (рисунок 21).

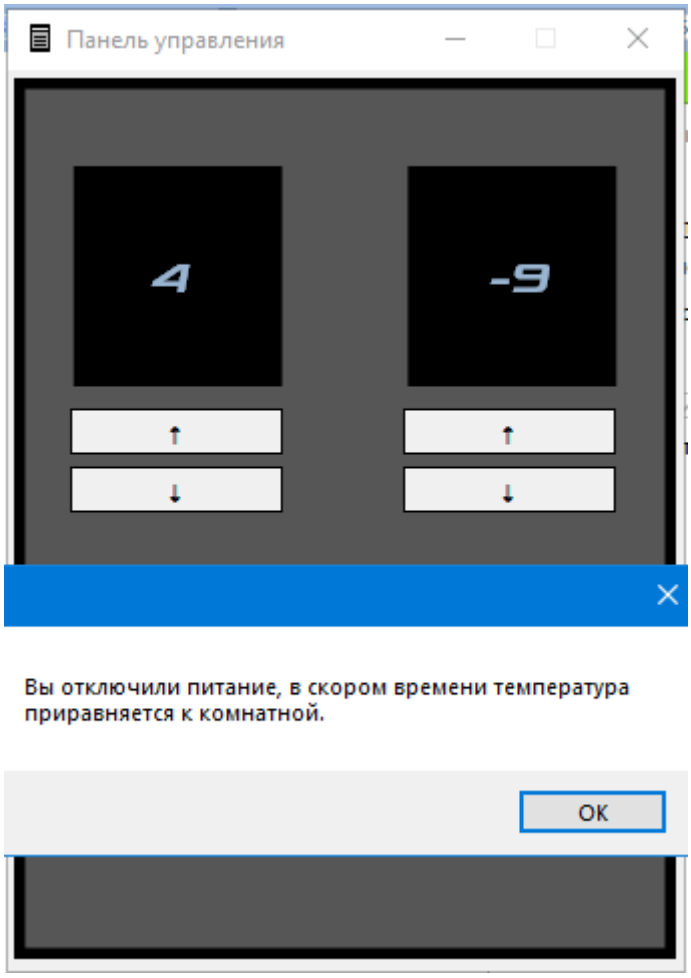


Рисунок 21 – Уведомление о действии

При смене режима, мы не только получаем уведомление, но параметры холодильника тоже изменяются. Меняется режим и температура, это все можно проследить на главной странице приложения, это отчетливо видно на рисунке 22.

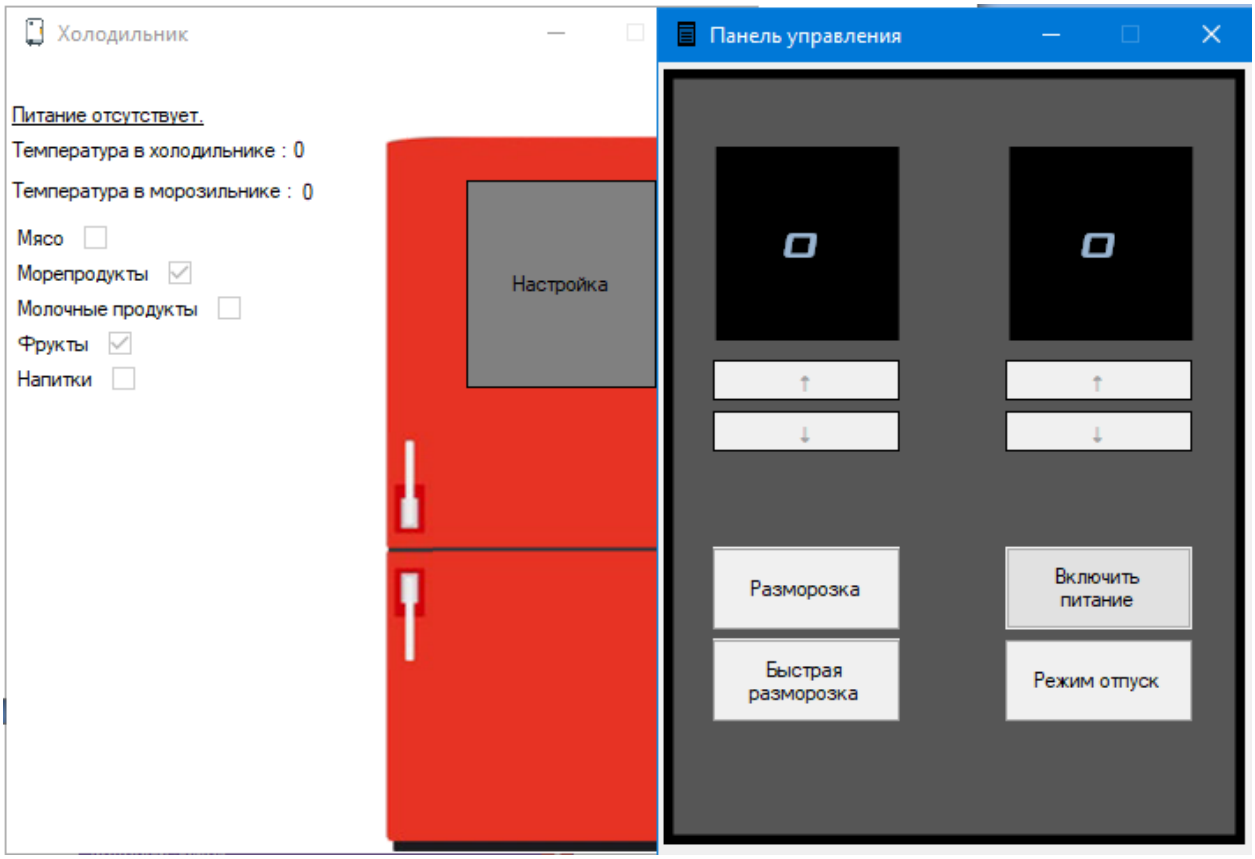


Рисунок 22 – Показание изменения параметров

Также в программе присутствует ещё одно окно, это окно открытого холодильника, в нем мы можем менять присутствующие продукты. Чтобы вызвать это окно необходимо нажать на ручку холодильника. Под каждым продуктом расположен элемент “checkBox”, который отвечает за его наличие в холодильнике, это видно на рисунке 23

Рисунок 23 – Открытый холодильник

При нажатии на “checkBox” любого продукта он появляется на заранее заданном месте, а также в главной панели появляется наличие категории этих продуктов, это хорошо продемонстрировано на картинках 24 и 25.

Рисунок 24 – Меняем выбор продуктов

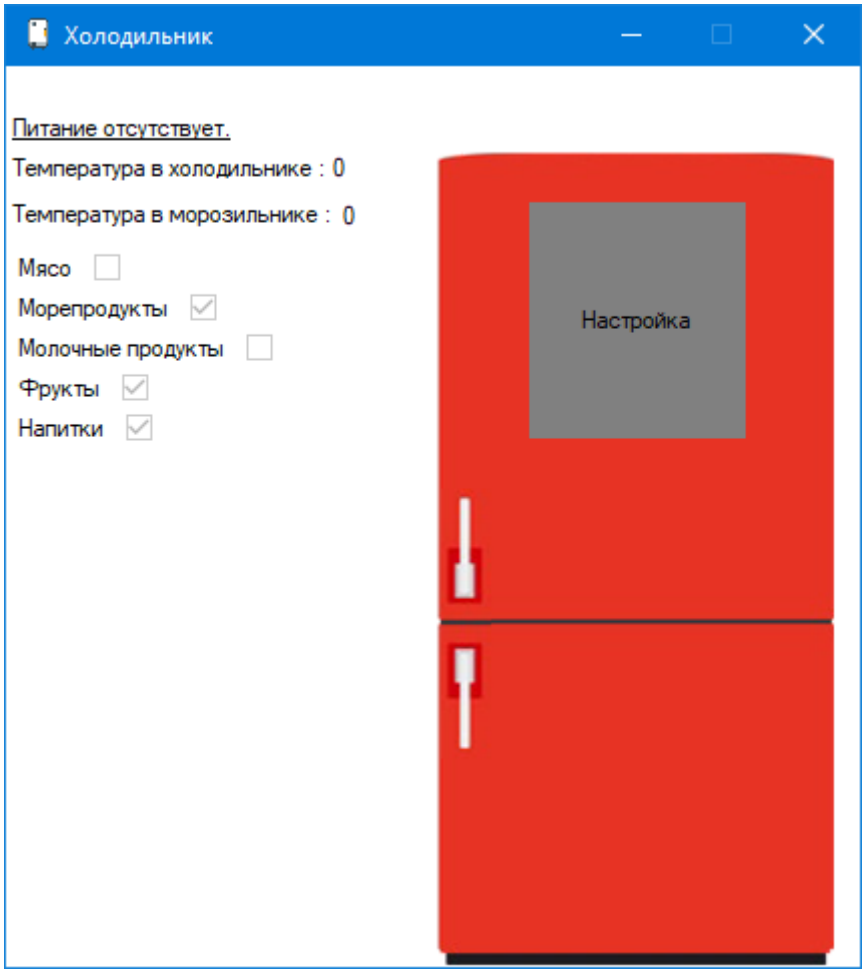


Рисунок 25 – Смотрим изменившуюся категорию продуктов

Ниже на рисунках 26, 27, 28 продемонстрирую оповещения о смене остальных режимов.

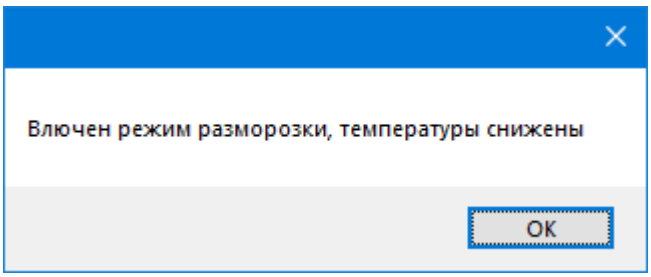


Рисунок 26 – Сообщение о режиме разморозки

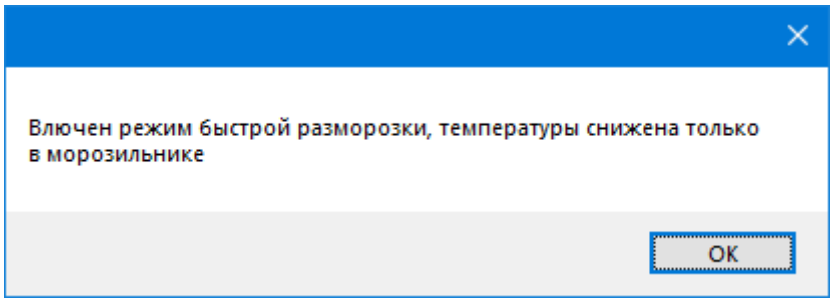


Рисунок 27 – Сообщение о режиме быстрой разморозки

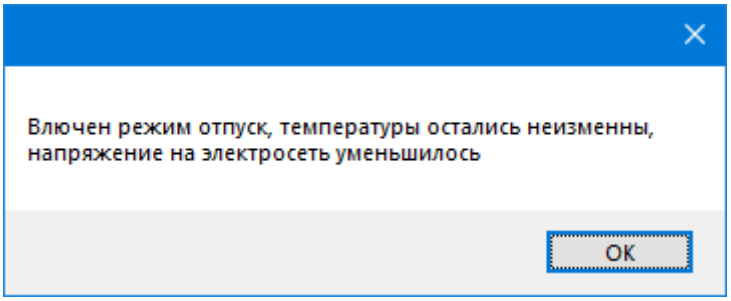


Рисунок 28 – Сообщение о режиме отпуск

9 Системные требования

Для оптимального функционирования приложения Холодильник необходима следующая минимальная конфигурация ПК:

- процессор имеющий 2 ядра и частоту не меньше 1,9ГГц;

- оперативную память объемом не менее 2ГБ;

- операционную систему Windows 7 или выше.

10 Руководство пользователя

Запуск программы можно осуществить следующим способом: открыть исполняемый файл.

Запуск программы при помощи исполняемого файла:

1) Найти исполняемый файл.
2) Запустить исполняемый файл с форматом .exe.

Теперь перед нами интерфейс программы: визуальный интерфейс, кнопка настроек холодильника, текстовые показатели и кнопка ручки открытия холодильника.

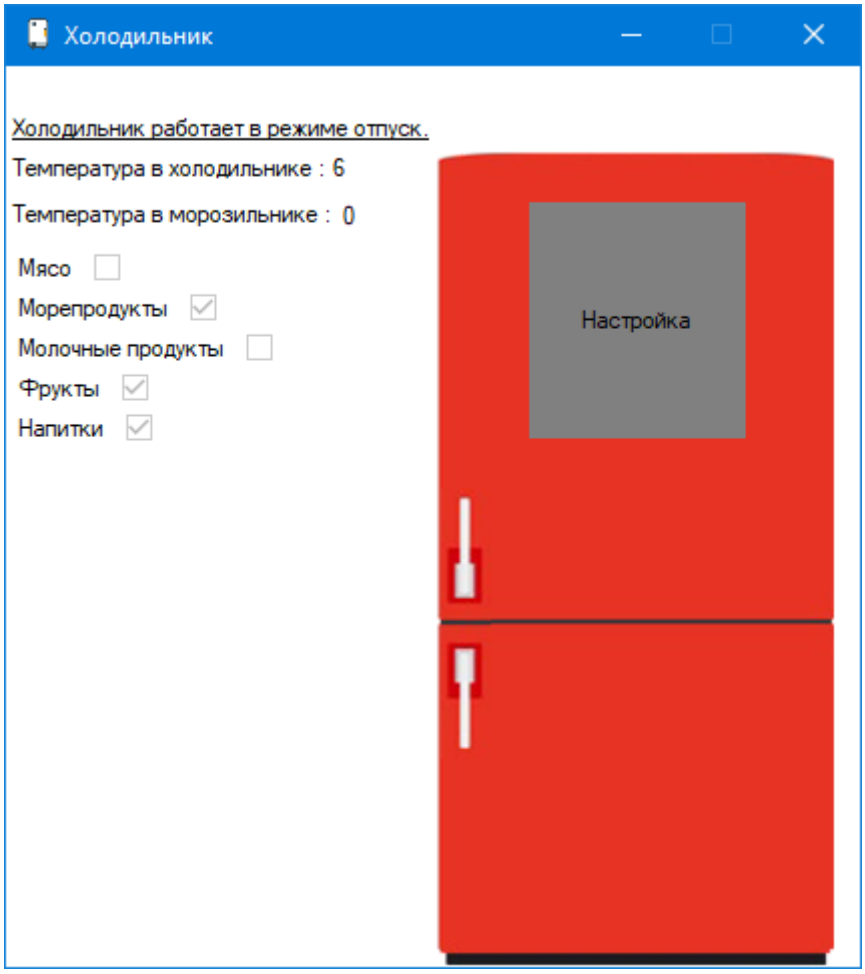


Рисунок 29 – Главное окно приложения

Пользователь имеет возможность пользоваться всем функционалом программы. Мы имеем два интерактивных элемента, кнопки настроек и кнопку открытия двери.

При нажатии на кнопку настроек, пользователю открывается интерфейс выбора режимов, весь список режимов:

- Разморозка;
- Быстрая разморозка;
- Отпуск;
- Отключение питания.

Выбирая любой из режимов, он будет применятся к холодильнику и менять параметры. При успешной смене режима, пользователь получит сообщение.

При нажатии кнопку ручки холодильника, перед нами открывается графический интерфейс открытого холодильника, в нем нам доступен ассортимент продуктов, которые мы можем вкладывать из выкладывать из него. При нажатие на поле под продуктом, он появится на заданном для него месте, а также на главном экране он прибавится в категорию.

Заключение

В результате выполнения данного курсового проекта была с нуля разработан проект «Холодильник». Мной была спроектирована программа, для нее был разработан функционал и интерфейс. Были построены несколько диаграм IDEF0, диаграмма IDEF3, также диаграмма UML, была проведена оценка трудозатрат, совершена планировка работ и как следствие диаграмма Ганта и сетевая диаграмма.

Была спроектирована система программного эмулирования работоспособности холодильника, симулирующую работы реального холодильника на языке высокого уровня C#, позволяющая наглядно продемонстрировать работу всех её компонентов. Полученные диаграммы позволяют детально изучить не только процесс машинного выполнения программы, но также и оценить процесс создания (проектирования и реализации) данного проекта.

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 полученный холодильник, эмулирующий работу реального устройства полностью позволяет протестировать работу будущего проекта, для которого он был спроектирован.

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. Попова О.Б. Теория разработки программного обеспечения. Методические указания по выполнению лабораторных работ для студентов всех форм обучения направления подготовки 09.03.04 Программная инженерия. Краснодар, 2019 – 69 с.

2. Попова О.Б. Теория разработки программного обеспечения. Конспект лекций.

3. Git за полчаса [электронный ресурс] URL: https://proglib.io/p/git-for-half-an-hour (дата обращения 13.03.2021)

4. Буч Г., Рамбо Дж., Джекобсон А. Язык UML. Руководство пользователя. – С-П.: Издательство «Питер», 2003. – 432 с.

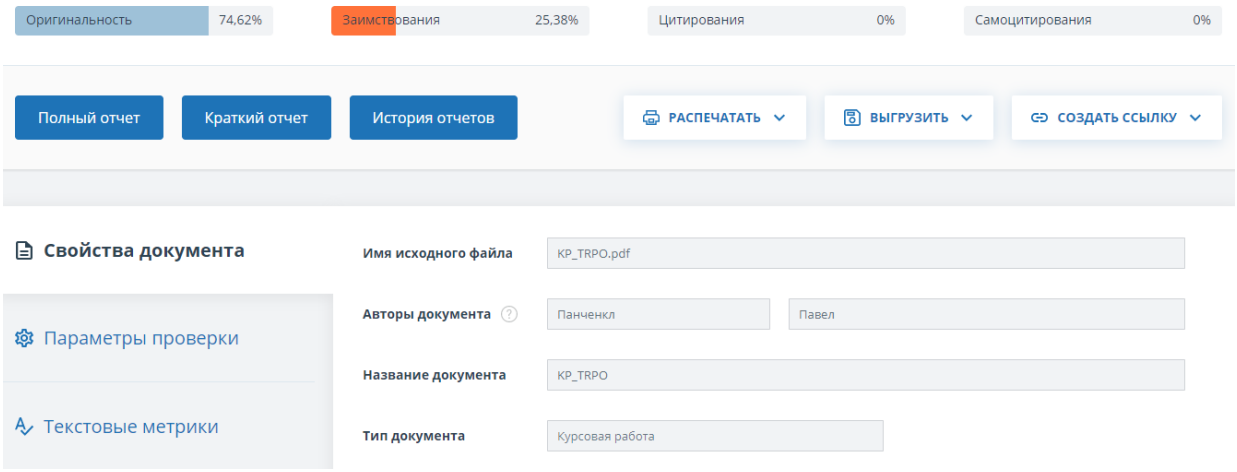
5. Ларман К. Применение UML и шаблонов проектирования: Введение в объектно-ориентированный анализ и проектирование: Учебное пособие: Пер. с англ. - М.: Вильямс, 2001. - 496 с.

6. Джепикс Филипп, Троелсен Эндрю. Объектно-ориентированный анализ и проектирование систем: Вильясм, 2018. - 1328 с.

7. Методы оценки – функциональные точки [электронный ресурс] URL: https://coderlessons.com/tutorials/akademicheskii/izuchite-metody-otsenki/metody-otsenki-funktsionalnye-tochki (дата обращения 15.03.2021)

Приложение А

Проверка на плагиат



Приложение Б

Листинг программы

```
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;

namespace Kyrs_Project
{
    public partial class Form1 : Form
    {
        public string Txt
        {
            get { return label1.Text; }
            set { label1.Text = value; }
        }
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            panel_ypr form2 = new panel_ypr();
            form2.ShowDialog();
        }

        private void button2_Click(object sender, EventArgs e)
        {
            open form1 = new open();
            form1.Show();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            label1.Text = Convert.ToString(label1.Text);
        }


        private void Form1_Activated(object sender, EventArgs e)
        {
            label1.Text = Properties.Settings.Default.tempN;
            label2.Text = Properties.Settings.Default.tempM;
            label10.Text = Properties.Settings.Default.HMode;
```

```
            if (Properties.Settings.Default.Meet == 0 &
Properties.Settings.Default.Chicken == 0) checkBox1.Checked =
false;
            else checkBox1.Checked = true;
            if (Properties.Settings.Default.Fish == 0 &
Properties.Settings.Default.Ant == 0 &
Properties.Settings.Default.BigFish == 0  &
Properties.Settings.Default.Tentackle == 0) checkBox2.Checked =
false;
            else checkBox2.Checked = true;
            if (Properties.Settings.Default.Milk == 0 &
Properties.Settings.Default.Cip == 0) checkBox3.Checked = false;
            else checkBox3.Checked = true;
            if (Properties.Settings.Default.Melon == 0 &
Properties.Settings.Default.Pineapple == 0 &
Properties.Settings.Default.Vine == 0 &
Properties.Settings.Default.Vegan == 0) checkBox4.Checked =
false;
            else checkBox4.Checked = true;
            if (Properties.Settings.Default.Soda == 0 &
Properties.Settings.Default.TomatoSouce == 0 &
Properties.Settings.Default.Yksys == 0 &
Properties.Settings.Default.Juice == 0) checkBox5.Checked =
false;
            else checkBox5.Checked = true;
        }


    }
}
```

```
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;

namespace Kyrs_Project
{
    public partial class open : Form
    {
        public open()
        {
            InitializeComponent();
        }

        private void pictureBox19_Click(object sender, EventArgs
e)
        {
```

```
        }

        private void open_Load(object sender, EventArgs e)
        {
            pictureBox2.BackColor = Color.Transparent;
            pictureBox16.Visible = false;
            pictureBox17.Visible = false;
            pictureBox18.Visible = false;
            pictureBox19.Visible = false;
            pictureBox22.Visible = false;
            pictureBox23.Visible = false;
            pictureBox24.Visible = false;
            pictureBox25.Visible = false;
            pictureBox26.Visible = false;
            pictureBox27.Visible = false;
            pictureBox28.Visible = false;
            pictureBox29.Visible = false;
            pictureBox30.Visible = false;
            pictureBox31.Visible = false;
            pictureBox32.Visible = false;
            pictureBox33.Visible = false;


        }

        private void pictureBox18_Click(object sender, EventArgs
e)
        {

        }

        private void checkBox1_CheckedChanged(object sender,
EventArgs e)
        {
            if (checkBox1.Checked) { pictureBox16.Visible =
true; Properties.Settings.Default.Meet++; }
            else {pictureBox16.Visible = false;
Properties.Settings.Default.Meet=0; }
        }

        private void checkBox2_CheckedChanged(object sender,
EventArgs e)
        {
            if (checkBox2.Checked) { pictureBox17.Visible =
true; Properties.Settings.Default.Fish++; }
            else { pictureBox17.Visible = false;
Properties.Settings.Default.Fish = 0; }
        }

        private void checkBox3_CheckedChanged(object sender,
EventArgs e)
        {
```

```
            if (checkBox3.Checked) { pictureBox28.Visible =
true; Properties.Settings.Default.Chicken++; }
            else { pictureBox28.Visible = false;
Properties.Settings.Default.Chicken = 0; }
        }

        private void checkBox4_CheckedChanged(object sender,
EventArgs e)
        {
            if (checkBox4.Checked) { pictureBox25.Visible =
true; Properties.Settings.Default.Ant++; }
            else { pictureBox25.Visible = false;
Properties.Settings.Default.Ant = 0; }
        }

        private void checkBox5_CheckedChanged(object sender,
EventArgs e)
        {
            if (checkBox5.Checked) { pictureBox26.Visible =
true; Properties.Settings.Default.Tentackle++; }
            else { pictureBox26.Visible = false;
Properties.Settings.Default.Tentackle = 0; }
        }

        private void checkBox6_CheckedChanged(object sender,
EventArgs e)
        {
            if (checkBox6.Checked) { pictureBox27.Visible =
true; Properties.Settings.Default.BigFish++; }
            else { pictureBox27.Visible = false;
Properties.Settings.Default.BigFish = 0; }
        }

        private void checkBox7_CheckedChanged(object sender,
EventArgs e)
        {
            if (checkBox7.Checked) { pictureBox29.Visible =
true; Properties.Settings.Default.Juice++; }
            else { pictureBox29.Visible = false;
Properties.Settings.Default.Juice = 0; }
        }

        private void checkBox8_CheckedChanged(object sender,
EventArgs e)
        {
            if (checkBox8.Checked) { pictureBox30.Visible =
true; Properties.Settings.Default.Milk++; }
            else { pictureBox30.Visible = false;
Properties.Settings.Default.Milk = 0; }
        }

        private void checkBox9_CheckedChanged(object sender,
EventArgs e)
```

```
        {
            if (checkBox9.Checked) { pictureBox31.Visible =
true; Properties.Settings.Default.Yksys++; }
            else { pictureBox31.Visible = false;
Properties.Settings.Default.Yksys = 0; }
        }

        private void checkBox10_CheckedChanged(object sender,
EventArgs e)
        {
            if (checkBox10.Checked) { pictureBox32.Visible =
true; Properties.Settings.Default.TomatoSouce++; }
            else { pictureBox32.Visible = false;
Properties.Settings.Default.TomatoSouce = 0; }
        }

        private void checkBox11_CheckedChanged(object sender,
EventArgs e)
        {
            if (checkBox11.Checked) { pictureBox33.Visible =
true; Properties.Settings.Default.Soda++; }
            else { pictureBox33.Visible = false;
Properties.Settings.Default.Soda = 0; }
        }

        private void checkBox12_CheckedChanged(object sender,
EventArgs e)
        {
            if (checkBox12.Checked) { pictureBox22.Visible =
true; Properties.Settings.Default.Melon++; }
            else { pictureBox22.Visible = false;
Properties.Settings.Default.Melon = 0; }
        }

        private void checkBox13_CheckedChanged(object sender,
EventArgs e)
        {
            if (checkBox13.Checked) { pictureBox19.Visible =
true; Properties.Settings.Default.Pineapple++; }
            else { pictureBox19.Visible = false;
Properties.Settings.Default.Pineapple = 0; }
        }

        private void checkBox14_CheckedChanged(object sender,
EventArgs e)
        {
            if (checkBox14.Checked) { pictureBox23.Visible =
true; Properties.Settings.Default.Vine++; }
            else {pictureBox23.Visible = false;
Properties.Settings.Default.Vine = 0; }
        }
```

```
        private void checkBox15_CheckedChanged(object sender,
EventArgs e)
        {
            if (checkBox15.Checked) { pictureBox24.Visible =
true; Properties.Settings.Default.Vegan++; }
            else { pictureBox24.Visible = false;
Properties.Settings.Default.Vegan = 0; }
        }

        private void checkBox16_CheckedChanged(object sender,
EventArgs e)
        {
            if (checkBox16.Checked) { pictureBox18.Visible =
true; Properties.Settings.Default.Cip++; }
            else { pictureBox18.Visible = false;
Properties.Settings.Default.Cip = 0; }
        }

        private void open_Activated(object sender, EventArgs e)
        {
            if (Properties.Settings.Default.Meet > 0)
checkBox1.Checked = true;
            else checkBox1.Checked = false;

            if (Properties.Settings.Default.Fish > 0)
checkBox2.Checked = true;
            else checkBox2.Checked = false;

            if (Properties.Settings.Default.Chicken > 0)
checkBox3.Checked = true;
            else checkBox3.Checked = false;

            if (Properties.Settings.Default.Ant > 0)
checkBox4.Checked = true;
            else checkBox4.Checked = false;

            if (Properties.Settings.Default.Tentackle > 0)
checkBox5.Checked = true;
            else checkBox5.Checked = false;

            if (Properties.Settings.Default.BigFish > 0)
checkBox6.Checked = true;
            else checkBox6.Checked = false;

            if (Properties.Settings.Default.Juice > 0)
checkBox7.Checked = true;
            else checkBox7.Checked = false;

            if (Properties.Settings.Default.Milk > 0)
checkBox8.Checked = true;
            else checkBox8.Checked = false;
```

```
            if (Properties.Settings.Default.Yksys > 0)
checkBox9.Checked = true;
            else checkBox9.Checked = false;

            if (Properties.Settings.Default.TomatoSouce > 0)
checkBox10.Checked = true;
            else checkBox10.Checked = false;

            if (Properties.Settings.Default.Soda > 0)
checkBox11.Checked = true;
            else checkBox11.Checked = false;

            if (Properties.Settings.Default.Melon > 0)
checkBox12.Checked = true;
            else checkBox12.Checked = false;

            if (Properties.Settings.Default.Pineapple > 0)
checkBox13.Checked = true;
            else checkBox13.Checked = false;

            if (Properties.Settings.Default.Vegan > 0)
checkBox15.Checked = true;
            else checkBox15.Checked = false;

            if (Properties.Settings.Default.Vine > 0)
checkBox14.Checked = true;
            else checkBox14.Checked = false;

            if (Properties.Settings.Default.Cip > 0)
checkBox16.Checked = true;
            else checkBox16.Checked = false;

        }

        private void open_Deactivate(object sender, EventArgs e)
        {
            Properties.Settings.Default.Save();
        }
    }

}
```

```
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;

namespace Kyrs_Project
```

```
{
    public partial class panel_ypr : Form
    {
        public panel_ypr()
        {
            InitializeComponent();
        }

        private void panel_ypr_Load(object sender, EventArgs e)
        {

        }

        private void button1_Click(object sender, EventArgs e)
        {
            int temp1plus = Convert.ToInt32(label1.Text);
            temp1plus++;
            label1.Text = Convert.ToString(temp1plus);
        }

        private void button2_Click(object sender, EventArgs e)
        {
            int temp1minus = Convert.ToInt32(label1.Text);
            temp1minus--;
            label1.Text = Convert.ToString(temp1minus);
        }

        private void button4_Click(object sender, EventArgs e)
        {
            int temp2plus = Convert.ToInt32(label2.Text);
            temp2plus++;
            label2.Text = Convert.ToString(temp2plus);
        }

        private void button3_Click(object sender, EventArgs e)
        {
            int temp2minus = Convert.ToInt32(label2.Text);
            temp2minus--;
            label2.Text = Convert.ToString(temp2minus);
        }

        private void panel_ypr_Deactivate(object sender,
EventArgs e)
        {
            int mN = Convert.ToInt32(label1.Text); int mM =
Convert.ToInt32(label2.Text);
            if (mN < -15 | mN > 20) {
MessageBox.Show("Температуры оказались больше доступных и были
возвращены к нулю."); label1.Text = "0";}
            if (mM < -23 | mM > 5) {
MessageBox.Show("Температуры оказались больше доступных и были
возвращены к нулю."); label2.Text = "0"; }
            Properties.Settings.Default.tempN = label1.Text;
```

```
            Properties.Settings.Default.tempM = label2.Text;
            Properties.Settings.Default.Save();
        }

        private void panel_ypr_Activated(object sender,
EventArgs e)
        {
            string prov = Properties.Settings.Default.proverka;
            if (prov == "1") { button10.Visible = false;
button11.Visible = true; button1.Enabled = false;
button2.Enabled = false; button3.Enabled = false;
button4.Enabled = false; }
            else { button10.Visible = true; button11.Visible =
false; }
            label1.Text = Properties.Settings.Default.tempN;
            label2.Text = Properties.Settings.Default.tempM;
        }

        private void button8_Click(object sender, EventArgs e)
        {
            try
            {
                Properties.Settings.Default.proverka = "0";
                MessageBox.Show("Влючен режим разморозки,
температуры снижены");
                Properties.Settings.Default.tempN = "12";
                Properties.Settings.Default.tempM = "0";
                //int specialError =
Convert.ToInt32(Properties.Settings.Default.tempM);
                //specialError /= 0;
                label1.Text = Properties.Settings.Default.tempN;
                label2.Text = Properties.Settings.Default.tempM;
                button1.Enabled = true; button2.Enabled = true;
button3.Enabled = true; button4.Enabled = true;
                Properties.Settings.Default.HMode = "Холодильник
работает в режиме разморозки.";
            }
            catch(Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }

        private void button10_Click(object sender, EventArgs e)
        {
            Properties.Settings.Default.proverka = "1";
            MessageBox.Show("Вы отключили питание, в скором
времени температура приравняется к комнатной.");
            Properties.Settings.Default.tempN = "0";
            Properties.Settings.Default.tempM = "0";
            button10.Visible = false;
            button11.Visible = true;
            label1.Text = Properties.Settings.Default.tempN;
```

```
            label2.Text = Properties.Settings.Default.tempM;
            button1.Enabled = false; button2.Enabled = false;
button3.Enabled = false; button4.Enabled = false;
            Properties.Settings.Default.HMode = "Питание
отсутствует.";
        }

        private void button9_Click(object sender, EventArgs e)
        {
            Properties.Settings.Default.proverka = "0";
            MessageBox.Show("Влючен режим отпуск, температуры
остались неизменны, напряжение на электросеть уменьшилось");
            label1.Text = Properties.Settings.Default.tempN;
            label2.Text = Properties.Settings.Default.tempM;
            button1.Enabled = true; button2.Enabled = true;
button3.Enabled = true; button4.Enabled = true;
            Properties.Settings.Default.HMode = "Холодильник
работает в режиме отпуск.";
        }

        private void button7_Click(object sender, EventArgs e)
        {
            Properties.Settings.Default.proverka = "0";
            MessageBox.Show("Влючен режим быстрой разморозки,
температуры снижена только в морозильнике");
            Properties.Settings.Default.tempN = "6";
            Properties.Settings.Default.tempM = "0";
            label1.Text = Properties.Settings.Default.tempN;
            label2.Text = Properties.Settings.Default.tempM;
            button1.Enabled = true; button2.Enabled = true;
button3.Enabled = true; button4.Enabled = true;
            Properties.Settings.Default.HMode = "Холодильник
работает в режиме быстрой разморозки.";
        }

        private void button11_Click(object sender, EventArgs e)
        {
            Properties.Settings.Default.proverka = "0";
            MessageBox.Show("Вы включили холодильник, в скором
времени температура приравняется к значениям на циферблате.");
            button10.Visible = true;
            button11.Visible = false;
            Properties.Settings.Default.tempN = "4";
            Properties.Settings.Default.tempM = "-11";
            label1.Text = Properties.Settings.Default.tempN;
            label2.Text = Properties.Settings.Default.tempM;
            button1.Enabled = true; button2.Enabled = true;
button3.Enabled = true; button4.Enabled = true;
            Properties.Settings.Default.HMode = "Холодильник
работает в обычном режиме";
        }

    }
```

```
}
```