Министерство образования Республики Беларусь
Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерного проектирования

Кафедра проектирования информационных компьютерных систем

Дисциплина "Объектно-ориентированное программирование"

К защите допустить:
Руководитель курсовой работы
старший преподаватель
кафедры
_____ А.В.Михалькевич
14.05.2024

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

на тему

Приложение «Погода» для IOS устройств

БГУИР КР 1-40 05 01-10 № 145 ПЗ

Студент (подпись студента) М. Е. Белов

Курсовая работа
представлена на проверку
14.05.2024
_____
(подпись студента)

Минск 2024

Реферат

БГУИР КР 1-40 05 01-10 № 145 ПЗ, гр. 814303

М. Е. Белов, Приложение «Погода» для IOS устройств, Минск: БГУИР - 2024.

Пояснительная записка 55243 с., 1 рис., 0 табл.

Ключевые слова: IOS-приложение, мобильное приложение, приложение "Погода"

Предмет Объектно-ориентированное программирование, А.В.Михалькевич

Предмет: создание мобильного IOS-приложения Объект: шаблоны проектирования, разработка пользовательского интерфейса. Цель: создание IOS-приложения "Погода" с использованием архитектурного шаблона проектирования MVC. Методология проведения работы: в процессе решения поставленных задач спроектирован и разработан простой и удобный интерфейс приложения, разработана логика приложения с использованием паттерна MVC. Результаты работы: разработано приложение с удобным функционалом. Интерфейс приложения был максимально упрощен и в меру информативен. Область применения результатов: удовлетворение пользователей приложения, нуждающихся в точной, а главное быстрой информации о погоде. Ссылка на онлайн-репозиторий GitHub: https://github.com/BelowMartin/WeatherApp

Subject: creating an IOS mobile app Object: design templates, user interface development. Goal: create an IOS Weather app using the MVC architectural design template. Methodology of work: in the process of solving the tasks, a simple and convenient application interface was designed and developed, and the application logic was developed using the MVC pattern. The results: the developed application is user-friendly functionality. The app interface has been simplified to the maximum and is reasonably informative. The scope of the results: the user satisfaction of the application requiring accurate, and most importantly quick weather information. Link to the github online repository: https://github.com/BelowMartin/WeatherApp

Содержание

Введение

В современном мире стало нормой, в любой момент знать, какая погода на любой части планеты. В этом нам помогает бескрайняя информационная паутина. Однако с появлением смартфонов это стало еще удобнее, ибо доступ к этой информации лежит прямо у нас в

кармане. Приложения для поиска информации о погоде стали неотъемлемой частью жизни современного человека. С помощью них люди могут заранее подготовить нужную им одежду для выхода на улицу, или “удачно” захватить с собой зонтик. Цель курсовой работы состоит в создании удобного приложения, подсказывающего пользователю, какая погода сейчас за его окном или на другой части света.

1 Описание проекта

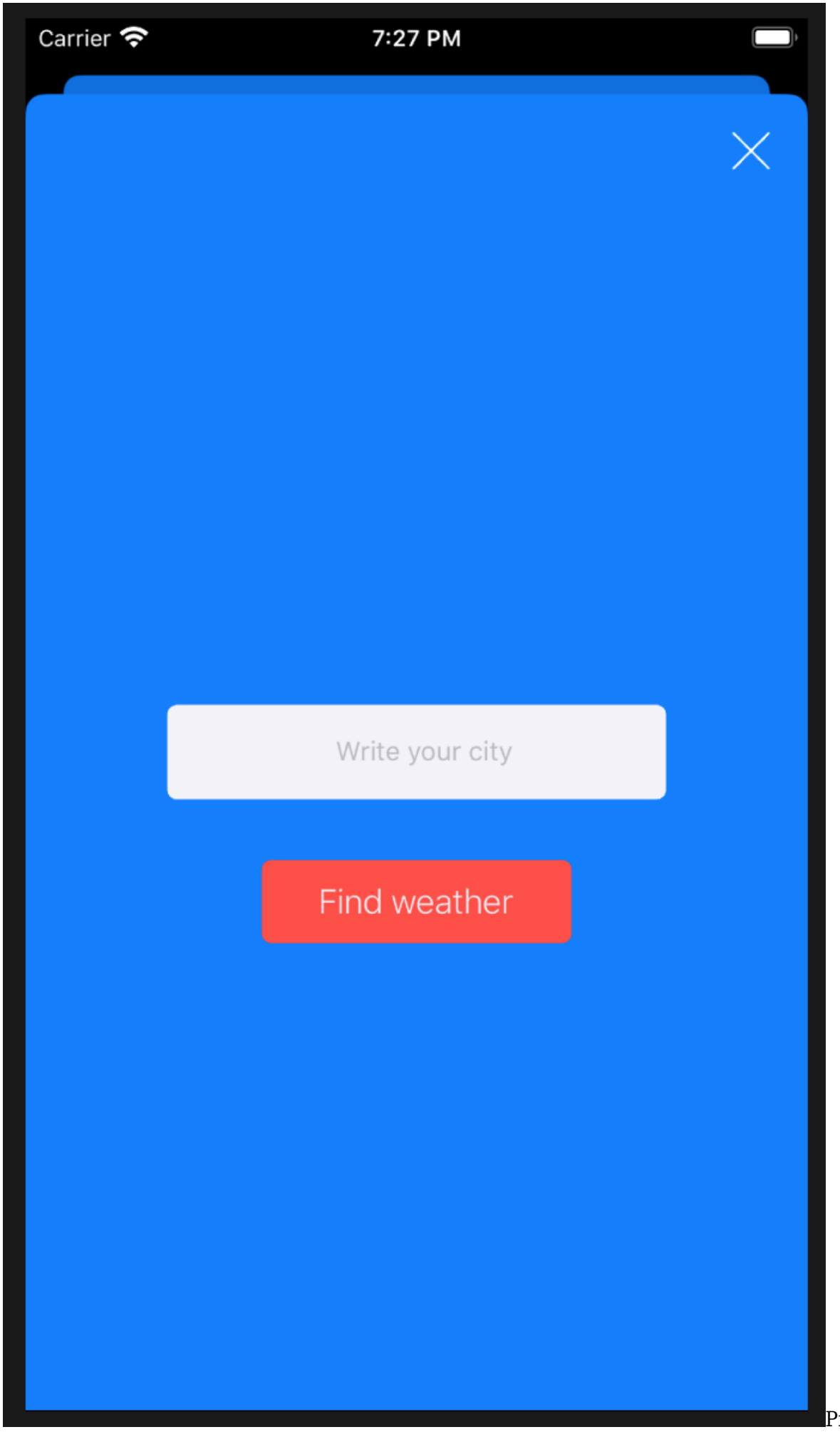

Рисунок - 2

При запуске приложения на экран отображается последний найденный результат поиска (рис. 1). На экране можно увидеть:

иконку, показывающую статус погоды в выбранном регионе
количество градусов по Цельсию для выбранном региона
название выбранного региона (города)
кнопку перехода на поисковой экран (в виде стрелки)

При нажатии на кнопку перехода, пользователь попадет на экран поиска нужного города (рис. 2). В поле поиска необходимо написать название искомого города. И после нажатия на кнопку «Find weather» пользователя перенесет на первый экран.

Полную версию программы можно найти на репозитории githubhttps://github.com/BelowMartin/WeatherApp

2 Выбранные технологии

Для разработки мобильных приложений компания Apple предлагает язык программирования Swift

Swift — открытый мультипарадигмальный компилируемый язык программирования общего назначения. Создан компанией Apple в первую очередь для разработчиков iOS и macOS. Swift работает с фреймворками Cocoa и Cocoa Touch и совместим с основной кодовой базой Apple, написанной на Objective-C. Swift задумывался как более лёгкий для чтения и устойчивый к ошибкам программиста язык, нежели предшествовавший ему Objective-C. Программы на Swift компилируются при помощи LLVM, входящей в интегрированную среду разработки Xcode 6 и выше. Swift может использовать рантайм Objective-C, что делает возможным использование обоих языков (а также C) в рамках одной программы.

Swift заимствовал довольно многое из Objective-C, однако он определяется не указателями, а типами переменных, которые обрабатывает компилятор. По аналогичному принципу работают многие скриптовые языки. В то же время, он предоставляет разработчикам многие функции, которые прежде были доступны в C++ и Java, такие как определяемые наименования, обобщения и перегрузка операторов.

Код, написанный на Swift, может работать вместе с кодом, написанным на языках программирования C и Objective-C в рамках одного и того же проекта[2].

Уже в экосистеме Swift были выбраны следующие Фреймворки для написания приложения: UIKit и Foundation для создания внешнего вида и логики перехода между экранами.

Основным и самым используемым Фреймворком в разработанном приложении послужил UIKit.

3 Инструментарий

Для написания программного кода под данное приложение использовалась официально поддерживаемая Apple среда разработки XCode 11.

Xcode — интегрированная среда разработки (IDE) программного обеспечения для платформ macOS, iOS, watchOS и tvOS, разработанная корпорацией Apple. Первая версия выпущена в 2003 году. Стабильные версии распространяются бесплатно через Mac App Store.

Пакет Xcode включает в себя изменённую версию свободного набора компиляторов GNU Compiler Collection и поддерживает языки C, C++, Objective-C, Objective-C++ (англ.)русск., Swift, Java, AppleScript, Python и Ruby с различными моделями программирования, включая (но не ограничиваясь) Cocoa, Carbon и Java. Сторонними разработчиками реализована поддержка GNU Pascal, Free Pascal, Ada, C#, Perl, Haskell и D. Пакет Xcode использует GDB в качестве back-end'а для своего отладчика.

Ну и наконец для сохранности кода и, в ходе разработки, возможности вернуться к предыдущей версии приложения в случае чего, использовалась система контроля версий Git.

Git — распределённая система управления версиями.

Среди проектов, использующих Git — ядро Linux, Swift, Android, Drupal, Cairo, GNU Core Utilities, Mesa, Wine, Chromium, Compiz Fusion, FlightGear, jQuery, PHP, NASM, MediaWiki, DokuWiki, Qt, ряд дистрибутивов Linux.

Система спроектирована как набор программ, специально разработанных с учётом их использования в сценариях. Это позволяет удобно создавать специализированные системы контроля версий на базе Git или пользовательские интерфейсы. Например, Cogito является именно таким примером оболочки к репозиториям Git, а StGit использует Git для управления коллекцией исправлений (патчей).

Git поддерживает быстрое разделение и слияние версий, включает инструменты для визуализации и навигации по нелинейной истории разработки. Как и Darcs, BitKeeper, Mercurial, Bazaar и Monotone, Git предоставляет каждому разработчику локальную копию всей истории разработки, изменения копируются из одного репозитория в другой.

Удалённый доступ к репозиториям Git обеспечивается git-демоном, SSH- или HTTP-сервером. TCP-сервис git-daemon входит в дистрибутив Git и является наряду с SSH наиболее распространённым и надёжным методом доступа. Метод доступа по HTTP, несмотря на ряд ограничений, очень популярен в контролируемых сетях, потому что позволяет использовать существующие конфигурации сетевых фильтров.

Ядро Git представляет собой набор утилит командной строки с параметрами. Все настройки хранятся в текстовых файлах конфигурации. Такая реализация делает Git легко портируемым на любую платформу и даёт возможность легко интегрировать Git в другие системы (в частности, создавать графические git-клиенты с любым желаемым интерфейсом).

Репозиторий Git представляет собой каталог файловой системы, в котором находятся файлы

конфигурации репозитория, файлы журналов, хранящие операции, выполняемые над репозиторием, индекс, описывающий расположение файлов, и хранилище, содержащее собственно файлы. Структура хранилища файлов не отражает реальную структуру хранящегося в репозитории файлового дерева, она ориентирована на повышение скорости выполнения операций с репозиторием. Когда ядро обрабатывает команду изменения (неважно, при локальных изменениях или при получении патча от другого узла), оно создаёт в хранилище новые файлы, соответствующие новым состояниям изменённых файлов. Существенно, что никакие операции не изменяют содержимого уже существующих в хранилище файлов.

По умолчанию репозиторий хранится в подкаталоге с названием «`.git`» в корневом каталоге рабочей копии дерева файлов, хранящегося в репозитории. Любое файловое дерево в системе можно превратить в репозиторий git, отдав команду создания репозитория из корневого каталога этого дерева (или указав корневой каталог в параметрах программы). Репозиторий может быть импортирован с другого узла, доступного по сети. При импорте нового репозитория автоматически создаётся рабочая копия, соответствующая последнему зафиксированному состоянию импортируемого репозитория (то есть не копируются изменения в рабочей копии исходного узла, для которых на том узле не была выполнена команда `commit`).

4 Архитектурный шаблон проектирования MVC

Model-View-Controller (**MVC**, «Модель-Представление-Контроллер», «Модель-Вид-Контроллер») — схема разделения данных приложения, пользовательского интерфейса и управляющей логики на три отдельных компонента: модель, представление и контроллер — таким образом, что модификация каждого компонента может осуществляться независимо.

- ***Модель*** (*Model*) предоставляет данные и реагирует на команды контроллера, изменяя своё состояние.
- ***Представление*** (*View*) отвечает за отображение данных модели пользователю, реагируя на изменения модели.
- ***Контроллер*** (*Controller*) интерпретирует действия пользователя, оповещая модель о необходимости изменений.

Основная цель применения этой концепции состоит в отделении бизнес-логики (*модели*) от её визуализации (*представления*, *вида*). За счёт такого разделения повышается возможность повторного использования кода. Наиболее полезно применение данной концепции в тех случаях, когда пользователь должен видеть те же самые данные одновременно в различных контекстах и/или с различных точек зрения. В частности, выполняются следующие задачи:

К одной *модели* можно присоединить несколько *видов*, при этом не затрагивая реализацию *модели*. Например, некоторые данные могут быть одновременно представлены в виде электронной таблицы, гистограммы и круговой диаграммы;

Не затрагивая реализацию *видов*, можно изменить реакции на действия пользователя (нажатие мышью на кнопке, ввод данных) — для этого достаточно использовать другой *контроллер*;

Ряд разработчиков специализируется только в одной из областей: либо разрабатывают графический интерфейс, либо разрабатывают бизнес-логику. Поэтому возможно добиться того, что программисты, занимающиеся разработкой бизнес-логики (*модели*), вообще не будут осведомлены о том, какое *представление* будет использоваться.

Концепция MVC позволяет разделить модель, представление и контроллер на три отдельных компонента:

Модель

Модель предоставляет данные и методы работы с ними: запросы в базу данных, проверка на корректность. Модель не зависит от представления (не знает, как данные визуализировать) и контроллера (не имеет точек взаимодействия с пользователем), просто предоставляя доступ к данным и управлению ими.

Модель строится таким образом, чтобы отвечать на запросы, изменяя своё состояние, при этом может быть встроено уведомление «наблюдателей».

Модель, за счёт независимости от визуального представления, может иметь несколько различных представлений для одной «модели».

Представление

Представление отвечает за получение необходимых данных из модели и отправляет их пользователю. Представление не обрабатывает введённые данные пользователя.

Контроллер

Контроллер обеспечивает «связь» между пользователем и системой. Контролирует и направляет данные от пользователя к системе и наоборот. Использует модель и представление для реализации необходимого действия.

Функциональные возможности и расхождения

Поскольку MVC не имеет строгой реализации, то реализован он может быть по-разному. Нет общепринятого определения, где должна располагаться бизнес-логика. Она может находиться как в контроллере, так и в модели. В последнем случае, модель будет содержать все бизнес-объекты со всеми данными и функциями.

Некоторые фреймворки жестко задают где должна располагаться бизнес-логика, другие не имеют таких правил.

Также не указано, где должна находиться проверка введённых пользователем данных. Простая валидация может встречаться даже в представлении, но чаще они встречаются в контроллере или модели.

Интернационализация и форматирование данных также не имеет четких указаний по расположению.

Схема алгоритма, диаграмма последовательности и диаграмма состояний приведены в Приложении пояснительной записки.

5 Шаблоны проектирования практических задач

Шаблон проектирования или паттерн (англ. *design pattern*) в разработке программного обеспечения — повторяемая архитектурная конструкция, представляющая собой решение проблемы проектирования в рамках некоторого часто возникающего контекста.

Обычно шаблон не является законченным образцом, который может быть прямо преобразован в код; это лишь пример решения задачи, который можно использовать в различных ситуациях. Объектно-ориентированные шаблоны показывают отношения и взаимодействия между классами или объектами, без определения того, какие конечные классы или объекты приложения будут использоваться.

«Низкоуровневые» шаблоны, учитывающие специфику конкретного языка программирования, называются идиомами. Это хорошие решения проектирования, характерные для конкретного языка или программной платформы, и потому не универсальные.

На наивысшем уровне существуют архитектурные шаблоны, они охватывают собой архитектуру всей программной системы.

Алгоритмы по своей сути также являются шаблонами, но не проектирования, а вычисления, так как решают вычислительные задачи.

Плюсы

В сравнении с полностью самостоятельным проектированием, шаблоны обладают рядом преимуществ. Основная польза от использования шаблонов состоит в снижении сложности разработки за счёт готовых абстракций для решения целого класса проблем. Шаблон даёт решению своё имя, что облегчает коммуникацию между разработчиками, позволяя ссылаться на известные шаблоны. Таким образом, за счёт шаблонов производится унификация деталей решений: модулей, элементов проекта, — снижается количество ошибок. Применение шаблонов концептуально сродни использованию готовых библиотек кода. Правильно сформулированный шаблон проектирования позволяет, отыскав удачное решение, пользоваться им снова и снова. Набор шаблонов помогает разработчику выбрать возможный, наиболее подходящий вариант проектирования.

Минусы

Хотя легкое изменение кода под известный шаблон может упростить понимание кода, по мнению Стива Макконнелла, с применением шаблонов могут быть связаны две сложности. Во-первых, слепое следование некоторому выбранному шаблону может привести к усложнению программы. Во-вторых, у разработчика может возникнуть желание попробовать некоторый шаблон в деле без особых оснований.

Многие шаблоны проектирования в объектно-ориентированном проектировании можно рассматривать как идиоматическое воспроизведение *элементов* функциональных языков. Питер Норвиг утверждает, что 16 из 23 шаблонов, описанных в книге «Банды четырёх», в динамически-типизируемых языках реализуются существенно проще, чем в C++, либо оказываются незаметны. Пол Грэхэм считает саму идею шаблонов проектирования — антипаттерном, сигналом о том, что система не обладает достаточным уровнем абстракции, и необходима её тщательная переработка. Нетрудно видеть, что само определение шаблона как «*готового решения, но не прямого обращения к библиотеке*» по сути означает отказ от повторного использования в пользу дублирования. Это, очевидно, может быть *неизбежным* для сложных систем при использовании языков, не поддерживающих комбинаторы и полиморфизм типов, и это в принципе может быть *исключено* в языках, обладающих свойством гомоиконичности (хотя и не обязательно эффективно), так как любой шаблон может быть реализован в виде исполнимого кода.

6 Приложение

GitHub: https://github.com/BelowMartin/WeatherApp

Заключение

Применение всех достоинств языка разработки, среды проектирования и необходимых Фреймворков, предоставленных компанией Apple, позволило качественно и быстро создать необходимое приложение. В ходе выполнения данной курсовой работы была достигнута поставленная цель: разработать красивое, удобное и быстрое приложение, которое позволит пользователю узнать погоду.

Список использованных источников

1. [url] **Swift** https://ru.wikipedia.org/wiki/Swift_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)
2. [url] **Objective – C** https://ru.wikipedia.org/wiki/Objective-C
3. [url] **Git** https://ru.wikipedia.org/wiki/Git
4. [url] **XCode** https://ru.wikipedia.org/wiki/Xcode
5. [url] **Сравнение Swift и Objective – C** https://geekbrains.ru/posts/swift_vs_obj_c
6. [url] **Шаблоны проектирования** https://ru.wikipedia.org/wiki/%D0%A8%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F
7. [url] **MVC** https://ru.wikipedia.org/wiki/Model-View-Controller

Приложения

1. [электронный документ] 5ebcfe897b809_Курсовая ООП.docx
2. [электронный документ] 5ebcfe99af37b_Курсовая ООП.docx
3. [электронный документ] 5ebd0174c2a94_рисунок 1.jpg
4. [электронный документ] 5ebd017a4111e_рисунок 2.jpg
5. [электронный документ] 5ebd051f057e0_Диаграммы.docx