Министерство образования Республики Беларусь

### Учреждение образования «[Институт информационных технологий](https://www.google.by/url?sa=t&rct=j&q=&esrc=s&source=web&cd=3&sqi=2&ved=0CDcQFjAC&url=http%3A%2F%2Fwww.iit-bsuir.by%2F&ei=unOvUpDmGemB4gSZ3IGQDQ&usg=AFQjCNHEr4the3QhkSSjmxbzcNJBZi5-Tg&sig2=GxV3hq7_8t34Csduk0HElg&bvm=bv.57967247,d.bGE&cad=rja) Белорусского государственного университета информатики и радиоэлектроники»

Факультет компьютерных технологий

Кафедра информационных систем и технологий

Дисциплина: Технологии разработки программного обеспечения

Контрольная работа

на тему

Автоматизация работы продавца магазина компьютерных комплектующих

Студент: гр. 581072 Годун А.В.

Руководитель: Бакунова О.М.

Минск 2017

# ВВЕДЕНИЕ

В современном обществе место, которое занимает торговый бизнес, невозможно переоценить. Каждый день любой из нас сталкивается с тем, что ему приходиться что-то покупать, или же, наоборот, сам продает или оказывает услуги. Постоянно мы вынуждены вступать в товарно-денежные отношения, часто даже не задумываясь о том, что это - наиболее распространенная форма взаимодействия между людьми.

Появление в 90-х гг. XX века электронной коммерции стало возможным благодаря развитию на планете сети Internet. Это, в свою очередь, было причиной возникновения и развития целых отраслей, связанных с обработкой информации. К примеру, деньги в наше время стали в представлении многих людей ассоциироваться не только, как бумажки с портретами и памятниками архитектуры, а с виртуальными счетами в интернет и обычных банках, с всевозможными платежными системами, кредитными картами, картами оплаты, балансами и т. д. Именно благодаря развитию информационного пространства, вовлечению в него миллионов пользователей, организаций и структур стало возможным появление в начале в США и Европе, а затем и в России и остальном мире, интернет-магазинов.

Особенно оказались они востребованы у так называемого среднего класса: людей, имеющих стабильную хорошо оплачиваемую работу и ценящих при этом своё время, удобство и комфорт. Все меньше в наши дни остается людей готовых тратить время на хождение по реальным магазинам, стояние в очередях и прочие неудобства. И все больше число тех, кто просто хочет кликнуть мышкой и потом лишь открыть входную дверь, чтобы получить то, что ему необходимо.

Выбранная мною тема считается актуальной на сегодняшний день, так как сегодня миллионы людей ежедневно, не выходя из дому, покупают различные товары в электронных магазинах. В мире, а в частности Беларуси огромными темпами растет количество пользователей internet и как следствие количество «электронных» покупателей, потенциальных «электронных» покупателей.

Электронные магазины существенно уменьшают издержки производителя, сэкономив на содержании обычного магазина, расширяют рынки сбыта, так же как и расширяет возможность покупателя - покупать любой товар в любое время в любой стране, в любом городе, в любое время суток, в любое время года. Это дает электронным магазинам неоспариваемое преимущество перед обычными магазинами. Этот момент является существенным при переходе производителей с «обычной» торговли на «электронную».

# 1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

## 1.1 Инструментальная поддержка процесса проектирования программного обеспечения

Инструментальная поддержка процесса проектирования программного обеспечения состоит из программных средств вычислительной техники, обеспечивающих автоматизацию процесса создания программного изделия.

Уровень автоматизации и инструментальной поддержки процесса проектирования программного обеспечения зависит от оснащенности процесса разработки и сопровождения программных изделий, которая включает в себя программные средства автоматизации технологических процессов разработки, изготовления и сопровождения программных изделий, а также аппаратные средства вычислительной техники, связи и тиражи­рования, используемые в типовой технологии.

Программная оснащенность разработчиков программного обеспечения определяется функциональными возможностями программных систем для автоматизации разработки программного обеспечения (САРПО) или иначе систем автоматизи­рованного проектирования программного обеспечения (САПР ПО).

Для каждого этапа разработки программного обеспечения могут применяться методы и средства, различающиеся эффективностью, которые, в частности, зависят от особенностей проекта.

В первом приближении степень программной оснащенности можно характеризовать объемом программ, активно используемых в типовой технологии.

Высокая степень программной оснащенности разработчиков достигается при автоматизации всех этапов разработки, изготовления и сопровождения программного изделия.

При этом используются следующие программные средства:

1) трансляторы программных спецификаций и текстов программ с языков высокого уровня;

2) средства планирования и контроля статического и динамического тестирования программ;

3) средства программного моделирования объектов внешней среды для программного продукта;

4) средства автоматизированного управления разработкой и конфигурационного контроля программных изделий;

5) средства современных методов автоматизации создания больших программных изделий.

Такие САПР ПО, объединяющие все эти методы и средства, имеют объемы до 1 млн. команд, и стоимость их разработки составляет несколько миллионов долларов. Однако средняя программная оснащенность разработчиков программ, как правило, значительно ниже.

Требования к системе автоматизированного проектирования ПО зависят от сложности объектов разработки, имеющихся ресурсов для создания программного обеспечения, и ряда других конструктивных и организационных факторов.

Требования к подобным системам для наиболее сложного

случая создания крупных программных средств состоят в следующем:

— снижение общей трудоемкости, длительности создания программного обеспечения;

— повышение производительности труда разработчиков;

— обеспечение высокого качества и надежности функционирования создаваемых и сопровождаемых программных продуктов;

— комплексная автоматизация коллективной разработки программного обеспечения большого объема и высокой сложности;

— обеспечение унифицированной технологии разработки и сопровождения программных изделий для реализующих их ЭВМ широкого класса;

— обеспечение эффективного использования ресурсов памяти и производительности реализующих ЭВМ.

Аппаратурная оснащенность разработчиков программного обеспечения определяется мощностью используемых профессиональных персональных ЭВМ и возможностью доступа к ним разработчиков программ.

# C++ представляет собой мощный универсальный язык программирования. Он может быть использован для создания небольших программ или больших приложений. С++ используется для написания CGI скриптов, также на С++ с лёгкостью создаются DOS программы. C++ позволяет создавать практически любые программы, какие только могут вам понадобиться.

## 1.2 Выбор инструментов разработки

Для запуска процесса установки системы поддержки учебного процесса, разработано оконное приложение в среде Microsoft Visual Studio 2008, у которого удобный, интуитивно понятный интерфейс для ввода необходимых параметров.

Visual Studio 2008 представляет собой полный набор средств, помогающих ускорить процесс реализации замысла разработчика. Это решение было создано, чтобы обеспечить поддержку проектов создания программного обеспечения для Интернета (включая ASP.NET AJAX), Windows Vista, Windows Server 2008, выпуска 2007 системы Microsoft Office, SQL Server 2008 и устройств под управлением Windows Mobile. Число платформ, на которые должны ориентироваться разработчики в соответствии с бизнес-требованиями, быстро увеличивается. Visual Studio 2008 Professional Edition предоставляет интегрированный набор средств, позволяющих учесть все эти требования путем расширения функциональности, доступной в Visual Studio 2008 Standard Edition.

Современным разработчикам приходится ориентироваться на широкий спектр платформ, создавая приложения, позволяющие организациям быстро получать ожидаемый результат. Встроенные в Visual Studio конструкторы и возможности языков программирования позволяют создавать приложения, способные связываться с удаленными базами данных и отвечающие чаяниям сегодняшнего бизнеса, а использование преимуществ среды .NET Framework 3.5 помогает сократить время разработки.

Из возможных языков программирования, предоставляемых Visual Studio 2008, для разработки визуальной оболочки был выбран C#.

C# - элегантный, строго типизированный объектно-ориентированный язык, предназначенный для разработки разнообразных безопасных и мощных приложений, выполняемых в среде .NET Framework. На языке C# можно разрабатывать обычные клиентские приложения Windows, веб-службы XML, распределенные компоненты, приложения типа “сервер-клиент”, приложения баз данных и многие-многие другие. В Visual C# 2008 имеется расширенный редактор кода, конструкторы с удобным пользовательским интерфейсом, встроенный отладчик и многие другие средства, призванные упростить разработку приложений на языке C# версии 3.0 и .NET Framework версии 3.5.

Синтаксис C# очень выразителен, но прост в изучении. Все, кто знаком с языками C, C++ или Java с легкостью узнают синтаксис с фигурными скобками, характерный для языка C#. Разработчики, знающие любой из этих языков, как правило, смогут добиться эффективной работы с языком C# за очень короткое время. Синтаксис C# делает проще то, что было сложно в C++, и обеспечивает мощные возможности, такие как типы значений Nullable, перечисления, делегаты, лямбда-выражения и прямой доступ к памяти, чего нет в Java. C# поддерживает универсальные методы и типы, обеспечивая более высокий уровень безопасности и производительности, а также итераторы, позволяющие при реализации коллекций классов определять собственное поведение итерации, которое может легко использоваться в клиентском коде. В C# 3.0 выражения LINQ (Language-Integrated Query) делают строго-типизированный запрос первоклассной конструкцией языка.

Как объектно-ориентированный язык, C# поддерживает понятия инкапсуляции, наследования и полиморфизма. Все переменные и методы, включая метод Main – точку входа приложения – инкапсулируются в определения классов. Класс может наследовать непосредственно из одного родительного класса, но может реализовывать любое число интерфейсов. Для методов, которые переопределяют виртуальные методы в родительском классе, необходимо ключевое слово override, чтобы исключить случайное повторное определение. В языке C# структура похожа на облегченный класс: это тип, распределяемый по стопкам, реализующий интерфейсы, но не поддерживающий наследование.

В дополнение к основным описанным объектно-ориентированным принципам, язык C# упрощает разработку компонентов программного обеспечения благодаря нескольким инновационным конструкциям языка, в число которых входят следующие:

- инкапсулированные подписи методов, называемые делегатами, которые поддерживают строго-типизированные уведомления о событиях;

- свойства, выступающие в роли методов доступа для закрытых переменных-членов;

- атрибуты с декларативными метаданными о типах во время выполнения;

- встроенные комментарии XML-документации;

- LINQ (Language-Integrated Query), предлагающий встроенные возможности запросов в различных источниках данных.

Если потребуется обеспечить взаимодействие с другим программным обеспечением Windows, таким как объекты COM или собственные библиотеки DLL Win32, в языке C# можно использовать процесс, который называется "Interop." Процесс Interop позволяет программам на C# выполнять практически любые действия, которые может выполнять исходное приложение на C++. Язык C# поддерживает даже указатели и понятие "небезопасного" кода для тех случаев, когда прямой доступ к памяти имеет крайне важное значение.

Процесс построения C# по сравнению с C и C++ прост и является более гибким, чем в Java. Нет отдельных файлов заголовка, а методы и типы не требуется объявлять в определенном порядке. В исходном файле C# может быть определено любое число классов, структур, интерфейсов и событий.

Основные действия по автоматизации установки реализованы с помощью утилиты NAnt. NAnt - консольная утилита, разработанная специально для работы с [.NET](http://ru.wikipedia.org/wiki/.NET_Framework) приложениями. Это свободное программное обеспечение для автоматизации процесса сборки программного обеспечения. Для работы утилиты необходимо наличие одной из версий [.NET Framework](http://ru.wikipedia.org/wiki/.NET_Framework). Управление процессом сборки происходит посредством специального конфигурационного [XML](http://ru.wikipedia.org/wiki/XML)-сценария, также называемого Build-файлом. В первую очередь этот файл содержит определение проекта, состоящего из отдельных целей (Targets). Цели сравнимы с [процедурами](http://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%B4%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0) в языках программирования и содержат вызовы команд-заданий (Tasks). Каждое задание представляет собой неделимую, атомарную команду, выполняющую некоторое элементарное действие.

Между целями могут быть определены зависимости — каждая цель выполняется только после того, как выполнены все цели, от которых она зависит (если они уже были выполнены ранее, повторного выполнения не производится).

Типичными примерами целей являются clean (удаление промежуточных файлов), compile ([компиляция](http://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%B8%D0%BB%D1%8F%D1%86%D0%B8%D1%8F) всех классов), deploy (развёртывание приложения на сервере). Конкретный набор целей и их взаимосвязи зависят от специфики проекта.

# 2 СИСТЕМНОЕ ПРОГРАММРОВАНИЕ

Моделирование деловых процессов, как правило, выполняется с помощью case-средств. К таким средствам относятся BPwin (PLATINUM technology), Silverrun (Silverrun technology), Oracle Designer (Oracle), Rational Rose (Rational Software) и др. Функциональные возможности инструментальных средств структурного моделирования деловых процессов будут рассмотрены на примере case-средства BPwin.

BPwin поддерживает три методологии моделирования: функциональное моделирование (IDEF0); описание бизнес-процессов (IDEF3); диаграммы потоков данных (DFD).

Функциональный блок, как отображающий моделируемую систему в целом (блок А0), так и блок на любом уровне декомпозиции являются преобразующими блоками. Преобразующий блок – блок IDEF0 – диаграммы, преобразующий входы в выходы под действием управлений при помощи «механизмов». Преобразование – цель и результат работы любого блока на диаграмме любого уровня декомпозиции. Преобразованию в блоке могут подвергаться материальные и информационные объекты, образующие соответствующие потоки. Материальный поток – непрерывное или дискретное множество материальных объектов, распределенное во времени. Информационный поток – множество информационных объектов, распределенное во времени. Информация, участвующая в процессах, операциях, действиях и деятельности в целом, может быть классифицирована на три группы: ограничительная информация; описательная информация; предписывающая (управляющая) информация. Ограничительная информация - сведения о том, чего нельзя делать:

а) никогда, ни при каких обстоятельствах (кроме, быть может, форс- мажорных) в любой фазе и на любом этапе функционирования системы в целом;

б) в рамках функционирования конкретного блока. Ограничительная информация содержится в законах, подзаконных актах, международных, государственных и отраслевых стандартах, а также в специальных внутренних положениях и документах предприятия, в частности, в технических требованиях, условиях, регламентах и т.д. Описательная информация – сведения об атрибутах объекта (потока) преобразуемого функциональным блоком. Содержится в чертежах, технических и иных описаниях, реквизитах и т.п. документах, являясь неотъемлемым компонентом объекта в течение всего жизненного цикла. Эта информация сама преобразуется (изменяется) в результате выполнения функции.

Предписывающая (управляющая) информация – сведения о том, как , при каких условиях и по каким правилам следует преобразовать объект (поток) на входе в объект (поток) на выходе блока. Содержится в технологических (в широком смысле) инструкциях, руководствах, документах, определяющих «настройки» и характеристики блока.

BPwin имеет достаточно простой и интуитивно понятный интерфейс пользователя. При запуске BPwin по умолчанию появляется основная панель инструментов, палитра инструментов (вид которой зависит от выбранной диаграммы) и, в левой части, навигатор модели — Model Explorer.

При создании новой модели возникает диалог, в котором следует указать, будет ли создана модель заново или она будет открыта из файла, затем внести имя модели и выбрать методологию, в которой будет построена модель.

Как было указано выше, BPwin поддерживает три методологии — IDEF0, IDEF3 и DFD, каждая из которых решает свои специфические задачи. В BPwin возможно построение смешанных моделей, т. е. модель может содержать одновременно диаграммы как IDEF0, так и IDEF3 и DFD. Состав палитры инструментов изменяется автоматически, когда происходит переключение с одной нотации на другую.

Модель в BPwin рассматривается как совокупность работ, каждая из которых оперирует с некоторым набором данных. Работа изображается в виде прямоугольников, данные — в виде стрелок. Если щелкнуть по любому объекту модели левой кнопкой мыши, появляется контекстное меню, каждый пункт которого соответствует редактору какого-либо свойства объекта.

## 2.1 Построение модели IDEF0

На начальных этапах создания ИС необходимо понять, как работает организация, которую собираются автоматизировать. Руководитель хорошо знает работу в целом, но не в состоянии вникнуть в детали работы каждого рядового сотрудника. Поэтому для описания работы предприятия необходимо построить модель, которая будет адекватна предметной области и содержать в себе знания всех участников бизнес-процессов организации.

Наиболее удобным языком моделирования бизнес-процессов является IDEF0, где система представляется как совокупность взаимодействующих работ или функций. Такая чисто функциональная ориентация является принципиальной — функции системы анализируются независимо от объектов, которыми они оперируют. Это позволяет более четко смоделировать логику и взаимодействие процессов организации.

Процесс моделирования системы в IDEF0 начинается с создания контекстной диаграммы — диаграммы наиболее абстрактного уровня описания системы в целом, содержащей определение субъекта моделирования, цели и точки зрения на модель.

Под субъектом понимается сама система, при этом необходимо точно установить, что входит в систему, а что лежит за ее пределами, другими словами, определить, что будет в дальнейшем рассматриваться как компоненты системы, а что как внешнее воздействие. На определение субъекта системы будут существенно влиять позиция, с которой рассматривается система, и цель моделирования – вопросы, на которые построенная модель должна дать ответ.

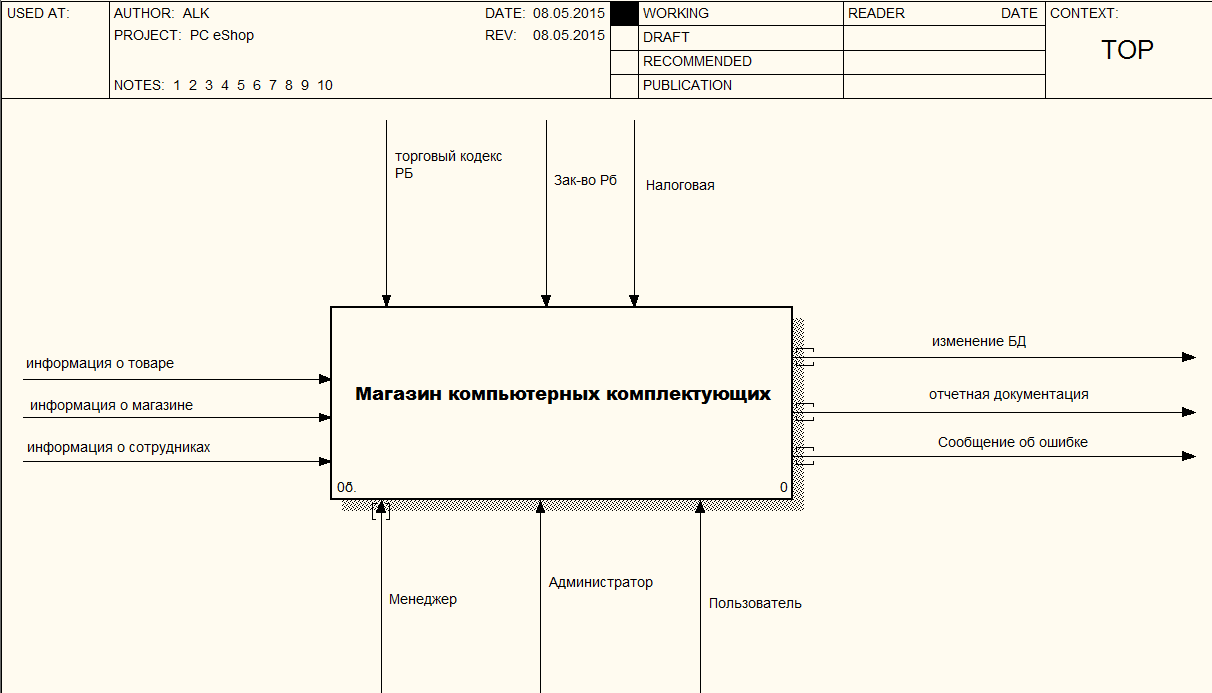


Рисунок 2.1 – Диаграмма A-0

На данной диаграмме отображены основные подпроцессы основного бизнес-процесса. В качестве входных потоков представлены:

* информация о товаре;
* информация о магазине;
* информация о сотрудниках.

В качестве выходных потоков представлены:

* отчетная документация;
* изменение БД;
* сообщение об ошибке.

Управляющими воздействиями являются такие, как торговый кодекс РБ, законодательство РБ, налоговая.

В качестве механизмов обозначены менеджер, администратор, пользователь. Процесс протекает в следующей последовательности:

* выбор товара;
* заказ товара;
* оплата товара;
* подтверждение оплаты;
* подготовка отчетной документации.

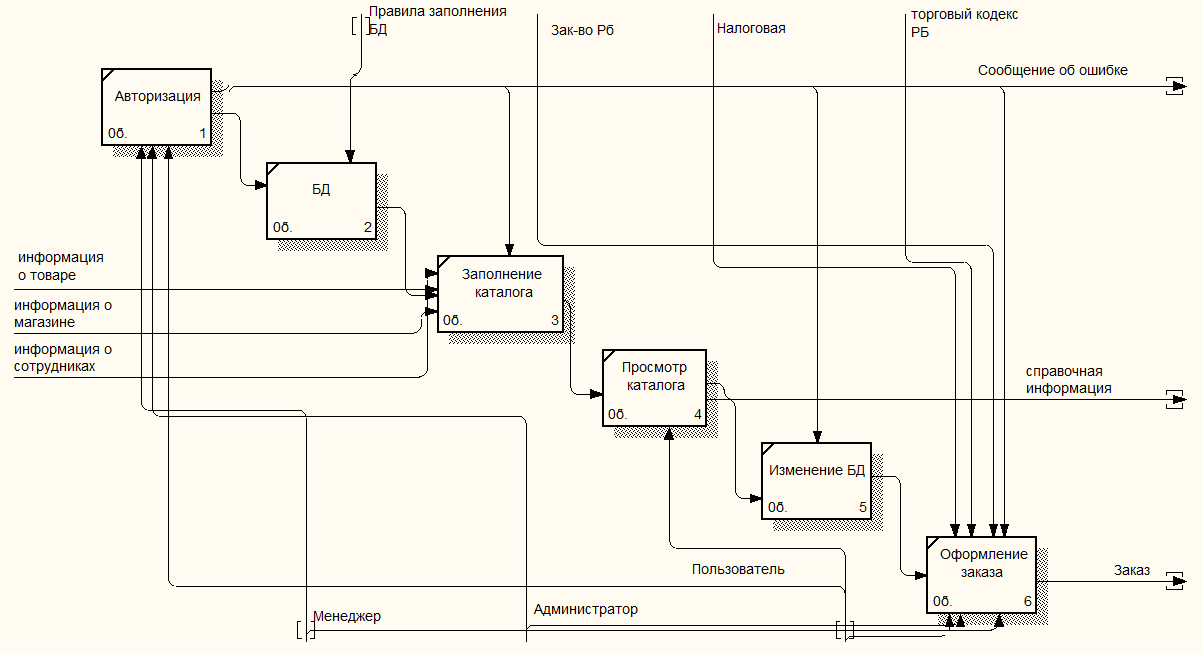


Рисунок 2.2 – Диаграмма A0

# 3 ФИЗИЧЕСКОЕ И ЛОГИЧЕСКОЕ МОДЕЛИРОВАНИЕ

Физическая модель приложения представлена в виде диаграммы классов (рисунок 3.1)

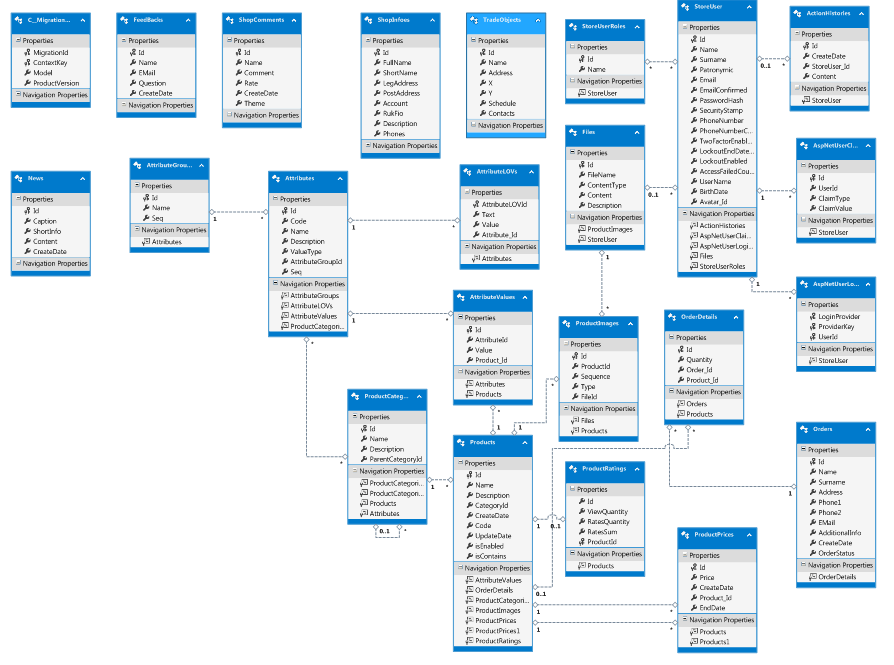


Рисунок 3.1 – Диаграмма классов

Назначения таблиц.

Таблица StoreUser хранит информацию о пользователях, структура приведена в таблице 3.1.

Таблица 3.1 – Структура таблицы StoreUser

|  |  |  |  |
| --- | --- | --- | --- |
| Название поля | Тип | Размер | Назначение |
| Username | String | 50 | Идентификация пользователя |
| Name | String | 50 | Имя |
| Surname | String | 50 | Фамилия |
| Patronymic | String | 50 | Отчество |
| BrthDate | DateTime |  | Дата рождения |
| Email | String | 100 | Email |
| PhoneNumber | String | 50 | Телефон |

Таблица Roles хранит информацию о ролях пользователей, структура приведена в таблице 3.2.

Таблица 3.2 – Структура таблицы Roles

|  |  |  |  |
| --- | --- | --- | --- |
| Название поля | Тип | Размер | Назначение |
| Id | Int |  | Идентификатор |
| Name | String | 100 | Название роли |

Таблица Products хранит общую информацию о товаре, структура приведена в таблице 3.3.

Таблица 3.3 – Структура таблицы Products

|  |  |  |  |
| --- | --- | --- | --- |
| Название поля | Тип | Размер | Назначение |
| Name | String | 100 | Название |
| Dscription | String | 1000 | Описание |
| CreateDate | DateTime |  | Дата cоздания |
| UpdateDate | DateTime |  | Дата изменения |
| CategoryId | Int |  | Ссылка на категорию |
| isEnable | Boolean |  | Доступность |

Таблица ProductCategory хранит описание категорий товаров, структура приведена в таблице 3.4.

Таблица 3.4 – Структура таблицы ProductCategory

|  |  |  |  |
| --- | --- | --- | --- |
| Название поля | Тип | Размер | Назначение |
| Name | String | 100 | Название |
| Description | String | 1000 | Описание |

Таблица Attributes хранить информацию об атрибутах товара, структура приведена в таблице 3.5.

Таблица 3.5 – Структура таблицы Attributes

|  |  |  |  |
| --- | --- | --- | --- |
| Название поля | Тип | Размер | Назначение |
| Name | String | 100 | Название |
| Description | String | 1000 | Описание |

Таблица AttributesValues хранит значения атрибутов для товара, структура приведена в таблице 3.6.

Таблица 3.6 – Структура таблицы AttributesValues

|  |  |  |  |
| --- | --- | --- | --- |
| Название поля | Тип | Размер | Назначение |
| ProductId | Int |  | Ссылка на продукт |
| AttribyteId | Int |  | Ссылка на атрибут |
| Value | String | 100 | Значение |

Таблица AttributesLOV хранит значения для выпадающих списков, структура приведена в таблице 3.7.

Таблица 3.7 – Структура таблицы AttributesLOV

|  |  |  |  |
| --- | --- | --- | --- |
| Название поля | Тип | Размер | Назначение |
| AttributeId | Int |  | Ссылка на атрибут |
| Text | String | 100 | Отображение |
| Value | String | 100 | Значение |

Таблица Order хранит информацию о заказе, структура приведена в таблице 3.8.

Таблица 3.8 – Структура таблицы Order

|  |  |  |  |
| --- | --- | --- | --- |
| Название поля | Тип | Размер | Назначение |
| Name | String | 100 | Имя заказчика |
| Surname | String | 100 | Фамилия заказчика |
| Phone | String | 100 | Телефон |
| Address | String | 500 | Адрес доставки |
| Status | Int |  | Статус заказа |
| CreateDate | DateTime |  | Дата создания |

Таблица OrderDetail хранит дополнительную информацию заказа, структура приведена в таблице 3.9.

Таблица 3.9 – Структура таблицы OrderDetail

|  |  |  |  |
| --- | --- | --- | --- |
| Название поля | Тип | Размер | Назначение |
| ProductId | Int |  | Ссылка на товар |
| Quantity | Int |  | Количество |

Таблица ShopInfo хранит информацию о магазине, структура приведена в таблице 3.10.

Таблица 3.10 – Структура таблицы OrderDetail

|  |  |  |  |
| --- | --- | --- | --- |
| Название поля | Тип | Размер | Назначение |
| FIO | String | 100 | ФИО рукововодителя |
| Name | String | 200 | Название организации |
| LegAddress | String | 500 | Юр. Адрес |
| Phones | String | 500 | Телефоны |
| Description | String | 2000 | Описание |

Разработка диаграммы вариантов использования преследует цели:

* Определить общие границы и контекст моделируемой предметной области на начальных этапах проектирования системы.
* Сформулировать общие требования к функциональному поведению проектируемой системы.
* Разработать исходную концептуальную модель системы для ее последующей детализации в форме логических и физических моделей.
* Подготовить исходную документацию для взаимодействия разработчиков системы с ее заказчиками и пользователями.

Суть данной диаграммы состоит в следующем: проектируемая система представляется в виде множества сущностей или актеров, взаимодействующих с системой с помощью так называемых вариантов использования. При этом актером (actor) или действующим лицом называется любая сущность, взаимодействующая с системой извне. Это может быть человек, техническое устройство, программа или любая другая система, которая может служить источником воздействия на моделируемую систему так, как определит сам разработчик. В свою очередь, вариант использования (use case) служит для описания сервисов, которые система предоставляет актеру. Другими словами, каждый вариант использования определяет некоторый набор действий, совершаемый системой при диалоге с актером. При этом ничего не говорится о том, каким образом будет реализовано взаимодействие актеров с системой.

В самом общем случае, диаграмма вариантов использования представляет собой граф специального вида, который является графической нотацией для представления конкретных вариантов использования, актеров, возможно некоторых интерфейсов, и отношений между этими элементами. При этом отдельные компоненты диаграммы могут быть заключены в прямоугольник, который обозначает проектируемую систему в целом. Следует отметить, что отношениями данного графа могут быть только некоторые фиксированные типы взаимосвязей между актерами и вариантами использования, которые в совокупности описывают сервисы или функциональные требования к моделируемой системе.

Между компонентами диаграммы вариантов использования могут существовать различные отношения, которые описывают взаимодействие экземпляров одних актеров и вариантов использования с экземплярами других актеров и вариантов. Один актер может взаимодействовать с несколькими вариантами использования. В этом случае этот актер обращается к нескольким сервисам данной системы. В свою очередь один вариант использования может взаимодействовать с несколькими актерами, предоставляя для всех них свой сервис. Следует заметить, что два варианта использования, определенные для одной и той же сущности, не могут взаимодействовать друг с другом, поскольку каждый из них самостоятельно описывает законченный вариант использования этой сущности. Более того, варианты использования всегда предусматривают некоторые сигналы или сообщения, когда взаимодействуют с актерами за пределами системы. В то же время могут быть определены другие способы для взаимодействия с элементами внутри системы.

В языке UML имеется несколько стандартных видов отношений между актерами и вариантами использования:

* + Отношение ассоциации (association relationship)
  + Отношение расширения (extend relationship)
  + Отношение обобщения (generalization relationship)
  + Отношение включения (include relationship)

При этом общие свойства вариантов использования могут быть представлены тремя различными способами, а именно с помощью отношений расширения, обобщения и включения.

Реализация варианта использования зависит от типа элемента модели, в котором он определен. Например, поскольку варианты использования класса определяются посредством операций этого класса, они реализуются соответствующими методами. С другой стороны, варианты использования подсистемы реализуются элементами, из которых состоит данная подсистема. Поскольку подсистема не имеет своего собственного поведения, все предлагаемые подсистемой сервисы должны представлять собой композицию сервисов, предлагаемых отдельными элементами этой подсистемы, т. е., в конечном итоге, классами. Эти элементы могут взаимодействовать друг с другом для совместного обеспечения требуемого поведения отдельного варианта использования. Такое совместное обеспечение требуемого поведения описывается специальным элементом языка UML - кооперация или сотрудничество. Отметим, что кооперации используются как для уточнения спецификаций в виде вариантов использования нижних уровней диаграммы, так и для описания особенностей их последующей реализации.

Если в качестве моделируемой сущности выступает система или подсистема самого верхнего уровня, то отдельные пользователи вариантов использования этой системы моделируются актерами. Такие актеры, являясь внутренними по отношению к моделируемым подсистемам нижних уровней, часто в явном виде не указываются, хотя и присутствуют неявно в модели подсистемы. Вместо этого варианты использования непосредственно обращаются к тем модельным элементам, которые содержат в себе подобные неявные актеры, т. е. экземпляры которых играют роли таких актеров при взаимодействии с вариантами использования. Эти модельные элементы могут содержаться в других пакетах или подсистемах. В последнем случае роли определяются в том пакете, к которому относится соответствующая подсистема.

Логическая модель представлена в диаграмме вариантов использования (рисунок 3.2)



Рисунок 3.2 – Диаграмма вариантов использования

Из диаграммы видно, что в приложении имеется 3 роли:

* Администратор
* Клиент
* Менеджер

Для моделирования процесса выполнения операций в языке UML используются так называемые диаграммы деятельности (Activitydiagram). Каждое состояние на диаграмме деятельности соответствует выполнению некоторой элементарной операции, а переход в следующее состояние срабатывает только при завершении этой операции в предыдущем состоянии. Графически диаграмма деятельности представляется в форме графа деятельности, вершинами которого являются состояния действия, а дугами – переходы от одного состояния действия к другому. На диаграмме деятельности отображается логика или последовательность перехода от одной деятельности к другой, при этом внимание фиксируется на результате деятельности. Сам же результат может привести к изменению состояния системы или возвращению некоторого значения. Диаграмма деятельности представлена на рисунке 3.3:

# 

Рисунок 3.3 – Диаграмма деятельности

[Диаграмма последовательности](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D0%B0%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0_%D0%BF%D0%BE%D1%81%D0%BB%D0%B5%D0%B4%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%BE%D1%81%D1%82%D0%B8) (Sequencediagram) - диаграмма, на которой показаны взаимодействия объектов, упорядоченные по времени их проявления, на которой изображено упорядоченное во времени взаимодействие объектов. В частности, на ней изображаются участвующие во взаимодействии объекты и последовательность сообщений, которыми они обмениваются. Диаграмма последовательности представлена на рисунке 3.4:



Рисунок 3.4 – Диаграмма последовательности

Физическое представление программной системы не может быть полным, если отсутствует информация о том, на какой платформе и на каких вычислительных средствах она реализована. Если разрабатывается программа, выполняющаяся локально на компьютере пользователя и не использующая периферийных устройств и ресурсов, то в разработке дополнительных диаграмм нет необходимости. При разработке же корпоративных приложений наличие таких диаграмм может быть крайне полезным для решения задач рационального размещения компонентов в целях эффективного использования распределенных вычислительных и коммуникационных ресурсов сети, обеспечения безопасности и других.

Для представления общей конфигурации и топологии распределенной программной системы в UML предназначены диаграммы развертывания.

Диаграмма развертывания предназначена для визуализации элементов и компонентов программы, существующих лишь на этапе ее исполнения (runtime). При этом представляются только компоненты-экземпляры программы, являющиеся исполняемыми файлами или динамическими библиотеками. Те компоненты, которые не используются на этапе исполнения, на диаграмме развертывания не показываются. Так, компоненты с исходными текстами программ могут присутствовать только на диаграмме компонентов. На диаграмме развертывания они не указываются.

Диаграмма развертывания содержит графические изображения процессоров, устройств, процессов и связей между ними. В отличие от диаграмм логического представления, диаграмма развертывания является единой для системы в целом, поскольку должна всецело отражать особенности ее реализации. Разработка диаграммы развертывания, как правило, является последним этапом спецификации модели программной системы.

При разработке диаграммы развертывания преследуют следующие цели:

* определить распределение компонентов системы по ее физическим узлам;
* показать физические связи между всеми узлами реализации системы на этапе ее исполнения;
* выявить узкие места системы и реконфигурировать ее топологию для достижения требуемой производительности.

Узел (node) представляет собой некоторый физически существующий элемент системы, обладающий определенным вычислительным ресурсом. В качестве вычислительного ресурса узла может рассматриваться наличие некоторого объема электронной или магнитооптической памяти или процессора. В последней версии языка UML понятие узла расширено и может включать в себя не только вычислительные устройства, но и другие механические или электронные устройства, такие как датчики, принтеры, модемы, цифровые камеры, сканеры и манипуляторы.

Графически на диаграмме развертывания узел изображается в форме трехмерного куба. Узел имеет собственное имя, которое указывается внутри его графического символа. Сами узлы могут представляться как в качестве типов, так и в качестве экземпляров.

В первом случае имя узла записывается без подчеркивания и начинается с заглавной буквы. Во втором - имя узла-экземпляра записывается в виде <имя узла ':' имя типа узла>. Имя типа узла указывает на некоторую разновидность узлов, присутствующих в модели системы.

Например, аппаратная часть системы может состоять из нескольких компьютеров, каждый из которых соответствует отдельному узлу-экземпляру в модели. Однако все эти узлы-экземпляры относятся к одному типу узлов, а именно узлу с именем типа «Компьютер».

Так же, как и на диаграмме компонентов, изображения узлов могут расширяться, чтобы включить некоторую дополнительную информацию о спецификации узла. Если дополнительная информация относится к имени узла, то она записывается под этим именем в форме помеченного значения.

Если необходимо явно указать компоненты, которые размещаются на отдельном узле, то это можно сделать двумя способами. Первый позволяет разделить графический символ узла на две секции горизонтальной линией. В верхней секции записывают имя узла, а в нижней размещенные на этом узле компоненты. Второй способ разрешает показывать на диаграмме развертывания узлы с вложенными изображениями компонентов. Однако нужно учитывать, что в качестве таких вложенных компонентов могут выступать только исполняемые компоненты.

В качестве дополнения к имени узла могут использоваться различные стереотипы, которые явно специфицируют назначение этого узла. Хотя в языке UML стереотипы для узлов не определены, в литературе встречаются следующие их варианты: «процессор», «датчик», «модем», «сеть» и другие, которые самостоятельно могут быть определены разработчиком. На диаграммах развертывания для различных физических устройств также допускаются специальные графические обозначения, поясняющие и раскрывающие назначение или выполняемые устройством функции.

Кроме изображений узлов на диаграмме развертывания указываются отношения между ними. В качестве отношений выступают физические соединения между узлами и зависимости между узлами и компонентами, изображения которых тоже могут присутствовать на диаграммах развертывания.

Соединения являются разновидностью ассоциации и изображаются отрезками линий без стрелок. Наличие такой линии указывает на необходимость организации физического канала для обмена информацией между соответствующими узлами. Характер соединения может быть дополнительно специфицирован примечанием, помеченным значением или ограничением.

Диаграммы развертывания могут иметь сложную структуру, включающую вложенные компоненты, интерфейсы и другие аппаратные устройства. Диаграмма развёртывания представлена на рисунке 3.5:

****

Рисунок 3.5 – Диаграмма развёртывания

Диаграмма компонентов, в отличие от ранее рассмотренных диаграмм, описывает особенности физического представления системы. Она позволяет определить архитектуру разрабатываемой системы, установив зависимости между программными компонентами, в роли которых может выступать исходный и исполняемый код. Основными графическими элементами диаграммы компонентов являются компоненты, интерфейсы и зависимости между ними. Диаграмма компонентов разрабатывается для следующих целей:

* визуализации общей структуры исходного кода программной системы;
* спецификации исполняемого варианта программной системы;
* обеспечения многократного использования отдельных фрагментов программного кода;
* представления концептуальной и физической схем баз данных.

В разработке диаграмм компонентов участвуют как системные аналитики и архитекторы, так и программисты. Диаграмма компонентов обеспечивает согласованный переход от логического представления к конкретной реализации проекта в форме программного кода. Одни компоненты могут существовать только на этапе компиляции программного кода, другие на этапе его исполнения. Диаграмма компонентов отражает общие зависимости между компонентами, рассматривая последние в качестве классификаторов. Выделяют три вида компонентов:

* развертывания, которые обеспечивают непосредственное выполнение системой своих функций. Такими компонентами могут быть динамически подключаемые библиотеки с расширением dll, Web-страницы на языке разметки гипертекста с расширением html и файлы справки с расширением hlp;
* рабочие продукты. Как правило, это файлы с исходными текстами программ, например, с расширениями h или срр для языка C++;
* исполнения, представляющие собой исполняемые модули - файлы с расширением ехе.

Эти элементы иногда называют артефактами, подчеркивая при этом их законченное информационное содержание, зависящее от конкретной технологии реализации соответствующих компонентов.

Другим способом спецификации различных видов компонентов является явное указание его стереотипа компонента перед именем. В языке UML для компонентов определены следующие стереотипы:

* библиотека (library) - определяет первую разновидность компонента, который представляется в форме динамической или статической библиотеки;
* таблица (table) - также определяет первую разновидность компонента, который представляется в форме таблицы базы данных;
* файл (file) - определяет вторую разновидность компонента, который представляется в виде файлов с исходными текстами программ;
* документ (document) - определяет вторую разновидность компонента, . который представляется в форме документа;
* исполнимый (executable) — определяет третий вид компонента, который может исполняться в узле.



Рисунок 3.6 – Диаграмма компонентов

Сгенерированный код:

# (object Petal

# version 50

# \_written "Rose 2006.0.0.060314"

# charSet 204)

# (object Design "Logical View"

# is\_unit TRUE

# is\_loaded TRUE

# attributes (list Attribute\_Set

# (object Attribute

# tool "Java"

# name "IDE"

# value "Internal Editor")

# (object Attribute

# tool "Java"

# name "UserDefineTagName1"

# value "")

# (object Attribute

# tool "Java"

# name "UserDefineTagText1"

# value "")

# (object Attribute

# tool "Java"

# name "UserDefineTagApply1"

# value "")

# (object Attribute

# tool "Java"

# name "UserDefineTagName2"

# value "")

# (object Attribute

# tool "Java"

# name "UserDefineTagText2"

# value "")

# (object Attribute

# tool "Java"

# name "UserDefineTagApply2"

# value "")

# (object Attribute

# tool "Java"

# name "UserDefineTagName3"

# value "")

# (object Attribute

# tool "Java"

# name "UserDefineTagText3"

# value "")

# (object Attribute

# tool "Java"

# name "UserDefineTagApply3"

# value ""))

# quid "36D487460258"

# enforceClosureAutoLoad FALSE

# defaults (object defaults

# rightMargin 0.250000

# leftMargin 0.250000

# topMargin 0.250000

# bottomMargin 0.500000

# pageOverlap 0.250000

# clipIconLabels TRUE

# autoResize TRUE

# snapToGrid TRUE

# gridX 16

# gridY 16

# defaultFont (object Font

# size 9

# face "Arial"

# charSet 204

# bold FALSE

# italics FALSE

# underline FALSE

# strike FALSE

# color 0

# default\_color TRUE)

# showMessageNum 1

# showClassOfObject TRUE

# notation "Unified")

# root\_usecase\_package (object Class\_Category "Use Case View"

# quid "34DBB4830142"

# documentation

# |The Use Case View organizes the Use-Case Model and the optional Business Use-Case Model.

# |

# 

# exportControl "Public"

# global TRUE

# logical\_models (list unit\_reference\_list

# (object Class "Пользователь"

# quid "554C01170297"

# stereotype "Actor")

# (object Class "Администратор"

# quid "554C01280287"

# stereotype "Actor")

# (object Class "Менеджер"

# quid "554C01320181"

# stereotype "Actor")

# (object UseCase "Поиск товара"

# quid "554C018102CA")

# (object UseCase "Просмотр информации о товаре"

# quid "554C019101CA")

# (object UseCase "Просмотр информации о магазине"

# quid "554C01A30171")

# (object UseCase "Обратная связь"

# quid "554C01B9015F")

# (object UseCase "Положить товар в корзину"

# quid "554C01F00348")

# (object UseCase "Оформить заказ"

# quid "554C02040321")

# (object UseCase "Авторизация"

# quid "554C022200E9")

# (object UseCase "Просмотр заказов"

# quid "554C024D03E7")

# (object UseCase "Изменить статус заказа"

# quid "554C02670271")

# (object UseCase "Ответить на обратную связь"

# quid "554C02850341")

# (object UseCase "Редактирование информации"

# quid "554C029A002A")

# (object UseCase "О пользователях"

# quid "554C02C90150")

# (object UseCase "О товарах"

# quid "554C02D501FB")

# (object UseCase "О магазине"

# quid "554C02FA012D")

# (object UseCase "Добавление новостей"

# quid "554C031C01D5")

# (object UseCase "Оставить комментарий к магазину"

# quid "554C032C001C")

# (object UseCase "NewUseCase"

# quid "554C035103B5")

# (object Association "$UNNAMED$0"

# quid "554C01DF0270"

# roles (list role\_list

# (object Role "$UNNAMED$1"

# quid "554C01E000CD"

# supplier "Use Case View::Обратная связь"

# quidu "554C01B9015F"

# is\_navigable TRUE)

# (object Role "$UNNAMED$2"

# quid "554C01E000D2"

# supplier "Use Case View::Пользователь"

# quidu "554C01170297")))

# (object Association "$UNNAMED$3"

# quid "554C01E502B1"

# roles (list role\_list

# (object Role "$UNNAMED$4"

# quid "554C01E60206"

# supplier "Use Case View::Просмотр информации о магазине"

# quidu "554C01A30171"

# is\_navigable TRUE)

# (object Role "$UNNAMED$5"

# quid "554C01E6020B"

# supplier "Use Case View::Пользователь"

# quidu "554C01170297")))

# (object Association "$UNNAMED$6"

# quid "554C01E8016E"

# roles (list role\_list

# (object Role "$UNNAMED$7"

# quid "554C01E90111"

# supplier "Use Case View::Просмотр информации о товаре"

# quidu "554C019101CA"

# is\_navigable TRUE)

# (object Role "$UNNAMED$8"

# quid "554C01E90113"

# supplier "Use Case View::Пользователь"

# quidu "554C01170297")))

# (object Association "$UNNAMED$9"

# quid "554C01EB014D"

# roles (list role\_list

# (object Role "$UNNAMED$10"

# quid "554C01EC0171"

# supplier "Use Case View::Поиск товара"

# quidu "554C018102CA"

# is\_navigable TRUE)

# (object Role "$UNNAMED$11"

# quid "554C01EC0173"

# supplier "Use Case View::Пользователь"

# quidu "554C01170297")))

# (object Association "$UNNAMED$12"

# quid "554C02150331"

# roles (list role\_list

# (object Role "$UNNAMED$13"

# quid "554C0216025A"

# supplier "Use Case View::Оформить заказ"

# quidu "554C02040321"

# is\_navigable TRUE)

# (object Role "$UNNAMED$14"

# quid "554C0216025C"

# supplier "Use Case View::Пользователь"

# quidu "554C01170297")))

# (object Association "$UNNAMED$15"

# quid "554C021901B2"

# roles (list role\_list

# (object Role "$UNNAMED$16"

# quid "554C021A0087"

# supplier "Use Case View::Положить товар в корзину"

# quidu "554C01F00348"

# is\_navigable TRUE)

# (object Role "$UNNAMED$17"

# quid "554C021A008D"

# supplier "Use Case View::Пользователь"

# quidu "554C01170297")))

# (object Association "$UNNAMED$18"

# quid "554C022C031C"

# roles (list role\_list

# (object Role "$UNNAMED$19"

# quid "554C022E0098"

# supplier "Use Case View::Авторизация"

# quidu "554C022200E9"

# is\_navigable TRUE)

# (object Role "$UNNAMED$20"

# quid "554C022E009D"

# supplier "Use Case View::Пользователь"

# quidu "554C01170297")))

# (object Association "$UNNAMED$21"

# quid "554C023300C2"

# roles (list role\_list

# (object Role "$UNNAMED$22"

# quid "554C02340145"

# supplier "Use Case View::Авторизация"

# quidu "554C022200E9"

# is\_navigable TRUE)

# (object Role "$UNNAMED$23"

# quid "554C0234014A"

# supplier "Use Case View::Менеджер"

# quidu "554C01320181")))

# (object Association "$UNNAMED$24"

# quid "554C023801F8"

# roles (list role\_list

# (object Role "$UNNAMED$25"

# quid "554C02390144"

# supplier "Use Case View::Авторизация"

# quidu "554C022200E9"

# is\_navigable TRUE)

# (object Role "$UNNAMED$26"

# quid "554C0239014E"

# supplier "Use Case View::Администратор"

# quidu "554C01280287")))

# (object Association "$UNNAMED$27"

# quid "554C023C01F9"

# roles (list role\_list

# (object Role "$UNNAMED$28"

# quid "554C023D01D7"

# supplier "Use Case View::Поиск товара"

# quidu "554C018102CA"

# is\_navigable TRUE)

# (object Role "$UNNAMED$29"

# quid "554C023D01E1"

# supplier "Use Case View::Менеджер"

# quidu "554C01320181")))

# (object Association "$UNNAMED$30"

# quid "554C024200AF"

# roles (list role\_list

# (object Role "$UNNAMED$31"

# quid "554C024300C8"

# supplier "Use Case View::Просмотр информации о товаре"

# quidu "554C019101CA"

# is\_navigable TRUE)

# (object Role "$UNNAMED$32"

# quid "554C024300CD"

# supplier "Use Case View::Менеджер"

# quidu "554C01320181")))

# (object Association "$UNNAMED$33"

# quid "554C02480117"

# roles (list role\_list

# (object Role "$UNNAMED$34"

# quid "554C0249012B"

# supplier "Use Case View::Оформить заказ"

# quidu "554C02040321"

# is\_navigable TRUE)

# (object Role "$UNNAMED$35"

# quid "554C02490130"

# supplier "Use Case View::Менеджер"

# quidu "554C01320181")))

# (object Association "$UNNAMED$36"

# quid "554C027502BC"

# roles (list role\_list

# (object Role "$UNNAMED$37"

# quid "554C02760100"

# supplier "Use Case View::Просмотр заказов"

# quidu "554C024D03E7"

# is\_navigable TRUE)

# (object Role "$UNNAMED$38"

# quid "554C02760106"

# supplier "Use Case View::Менеджер"

# quidu "554C01320181")))

# (object Association "$UNNAMED$39"

# quid "554C0279015E"

# roles (list role\_list

# (object Role "$UNNAMED$40"

# quid "554C027A0193"

# supplier "Use Case View::Изменить статус заказа"

# quidu "554C02670271"

# is\_navigable TRUE)

# (object Role "$UNNAMED$41"

# quid "554C027A0198"

# supplier "Use Case View::Менеджер"

# quidu "554C01320181")))

# (object Association "$UNNAMED$42"

# quid "554C0293032B"

# roles (list role\_list

# (object Role "$UNNAMED$43"

# quid "554C02940214"

# supplier "Use Case View::Ответить на обратную связь"

# quidu "554C02850341"

# is\_navigable TRUE)

# (object Role "$UNNAMED$44"

# quid "554C02940216"

# supplier "Use Case View::Менеджер"

# quidu "554C01320181")))

# (object Association "$UNNAMED$45"

# quid "554C02E80315"

# roles (list role\_list

# (object Role "$UNNAMED$46"

# quid "554C02E901E4"

# supplier "Use Case View::О пользователях"

# quidu "554C02C90150"

# is\_navigable TRUE)

# (object Role "$UNNAMED$47"

# quid "554C02E901E9"

# supplier "Use Case View::Редактирование информации"

# quidu "554C029A002A")))

# (object Association "$UNNAMED$48"

# quid "554C02EC0270"

# roles (list role\_list

# (object Role "$UNNAMED$49"

# quid "554C02ED014C"

# supplier "Use Case View::О товарах"

# quidu "554C02D501FB"

# is\_navigable TRUE)

# (object Role "$UNNAMED$50"

# quid "554C02ED014E"

# supplier "Use Case View::Редактирование информации"

# quidu "554C029A002A")))

# (object Association "$UNNAMED$51"

# quid "554C02EF0384"

# roles (list role\_list

# (object Role "$UNNAMED$52"

# quid "554C02F0028B"

# supplier "Use Case View::Редактирование информации"

# quidu "554C029A002A"

# is\_navigable TRUE)

# (object Role "$UNNAMED$53"

# quid "554C02F00290"

# supplier "Use Case View::Администратор"

# quidu "554C01280287")))

# (object Association "$UNNAMED$54"

# quid "554C0306009B"

# roles (list role\_list

# (object Role "$UNNAMED$55"

# quid "554C030603CD"

# supplier "Use Case View::О магазине"

# quidu "554C02FA012D"

# is\_navigable TRUE)

# (object Role "$UNNAMED$56"

# quid "554C030603CF"

# supplier "Use Case View::Редактирование информации"

# quidu "554C029A002A")))

# (object Association "$UNNAMED$57"

# quid "554C03270357"

# roles (list role\_list

# (object Role "$UNNAMED$58"

# quid "554C03280330"

# supplier "Use Case View::Добавление новостей"

# quidu "554C031C01D5"

# is\_navigable TRUE)

# (object Role "$UNNAMED$59"

# quid "554C03280335"

# supplier "Use Case View::Редактирование информации"

# quidu "554C029A002A")))

# (object Association "$UNNAMED$60"

# quid "554C0341002E"

# roles (list role\_list

# (object Role "$UNNAMED$61"

# quid "554C03410367"

# supplier "Use Case View::Оставить комментарий к магазину"

# quidu "554C032C001C"

# is\_navigable TRUE)

# (object Role "$UNNAMED$62"

# quid "554C03410369"

# supplier "Use Case View::Пользователь"

# quidu "554C01170297")))

# (object Class\_Category "Business Use-Case Model"

# quid "35B677D701B8"

# documentation

# |Optional Model:

# |The business modeling workflow in Rational Unified Process produces two models: the business use-case model, and the business object model. Both show the business

# |processes, but different aspects of them. In the business use-case model each business use case represents a business process, described (text and/or activity diagrams) from an "external" view point without worrying about who does what to whom inside of the organization.

# |In the business object model, you include realizations of each business use case to show how workers and entities collaborate to perform the process. You do that using class diagrams, activity diagrams with swimlanes, collaboration diagrams, and/or interaction diagrams.

# |

# |<Any additional description of the Business Use Case Model, such as modeling conventions that the reader/reviewer may need to understand the model, should be described here.>

# 

# exportControl "Public"

# logical\_models (list unit\_reference\_list)

# statemachine (object State\_Machine "State/Activity Model2"

# quid "554C12D6012C"

# states (list States)

# partitions (list Partitions)

# objects (list Objects)

# transitions (list transition\_list)

# objectflows (list objectflow\_list)

# statediagrams (list StateDiagrams

# (object ActivityDiagram "NewDiagram5"

# quid "554C12D602E5"

# title "NewDiagram5"

# zoom 100

# max\_height 28350

# max\_width 21600

# origin\_x 0

# origin\_y 0

# items (list diagram\_item\_list

# (object Swimlane "" @1

# line\_color 3342489

# width 450)))))

# logical\_presentations (list unit\_reference\_list

# (object UseCaseDiagram "Global View of Business Actors and Business Use Cases"

# quid "35C0DFF50172"

# title "Global View of Business Actors and Business Use Cases"

# documentation "This is a global view of the system showing all Busines Use Cases and Business Actors. It is intended to be an overview diagram, so if the diagram becomes too complex for easy comprehension, use several diagrams, each of which should focus on a different aspect of the system. Usually, organizing use case diagrams around actors is a good approach."

# zoom 100

# max\_height 28350

# max\_width 21600

# origin\_x 0

# origin\_y 0

# items (list diagram\_item\_list

# (object NoteView @2

# location (464, 352)

# font (object Font

# size 9

# face "Arial"

# charSet 204

# bold FALSE

# italics FALSE

# underline FALSE

# strike FALSE

# color 0

# default\_color TRUE)

# label (object ItemLabel

# Parent\_View @2

# location (42, 11)

# fill\_color 13434879

# nlines 14

# max\_width 809

# justify 1

# label

# |Global View of Business Actors and BusinessUse Cases

# |

# |This diagram presents all Business Actors and all concrete Business Use Cases (those that are directly instantiable).

# |

# |Related Rational Unified Process Activities:

# |- Activity: Find Business Actors and Use Cases

# |- Activity: Structure the Business Use Case Model

# |

# |\*Note: this note may be deleted once the diagram is created.

# )

# line\_color 3342489

# fill\_color 13434879

# width 869

# height 694)

# (object NoteView @3

# location (1280, 160)

# font (object Font

# size 9

# face "Arial"

# charSet 204

# bold FALSE

# italics FALSE

# underline FALSE

# strike FALSE

# color 0

# default\_color TRUE)

# label (object ItemLabel

# Parent\_View @3

# location (942, 29)

# fill\_color 13434879

# nlines 5

# max\_width 640

# justify 1

# label

# |Created/modified by: <name>

# |Date: <creation/modification date>

# |

# |Approved by: <name>

# |Date: <approval date>

# )

# line\_color 3342489

# fill\_color 13434879

# width 700

# height 275)))))

# (object Class\_Category "Use-Case Model"

# quid "35B677F4010E"

# documentation

# |The Use-Case Model in Rational Unified Process contains a model of the behavior of a system which supports the business processes.

# |<The description entered here should capture the Survey Description of the Use Case Model>.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Пряжников Н.С. Теория и практика профессионального самоопределения: Учеб. пособие. – М.: МГППИ, 1999. – 97 с.
2. Эндрю Р. CSS. 100 и 1 совет, 3-е изд.: Учеб. пособие. – СПб.: Символ-Плюс, 2010. – 336 с.
3. Климов А. JavaScript на примерах: Учеб. пособие. – СПб.: БХВ-Петербург, 2009. – 336 с.
4. Боггс, У. UML и Rational Rose 2002: Пер. с англ. / У. Боггс, М. Боггс. — М.: ЛОРИ, 2004
5. Дейтел Х.О, Дейтел П.О. Как программировать на С# - М.: Бином, 2005
6. Пол Нильсен. Microsoft SQL Server 2005. Библия пользователя.: Пер. с англ. – М.: ООО «И.Д. Вильямс», 2008
7. Макки А. Введение в .NET 4.0 и Visual studio 2010 для профессионалов. –СПБ.: Вильямс, 2010
8. ASP.NET MVC 5 с примерами на C# 5.0 для профессионалов. –СПБ.: Вильямс, 2014
9. Professional ASP.NET MVC 5. –СПБ.: Jon Galloway, Brad Wilson, K. Scott Allen, David Matson, 2014
10. CLR via C#. –СПБ.: Рихтер, 2014
11. Юрлов Ф.Ф. Технико-экономическая эффективность сложных радиоэлектронных систем. — М.: Сов. радио, 1980. — 280 с.
12. Юрлов Ф.Ф., Плеханова А.Ф., Маркитанов М.Ю. Выбор эффективных решений в экономике. — Н. Новгород: Изд-во НГТУ, 2004. — 115 с.
13. Сайт microsoft.com.
14. Гофман В., Хоменко А. Работа с базами данных в Delphi – Спб.: БХВ-Петербург, 2000. – 643 с.