МИНИСТЕРСТВО ОБРАЗОВАНИЯ МОСКОВСКОЙ ОБЛАСТИ
Государственное бюджетное профессиональное образовательное учреждение Московской области
«Воскресенский колледж»

МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ

ПМ 02. «Осуществление интеграции программных модулей»

Наименование специальности

09.02.07 «Информационные системы и программирование»

Квалификация выпускника

Программист

2020г.

Методические рекомендации по выполнению лабораторных работ по профессиональному модулю разработаны на основе Федерального государственного образовательного стандарта (далее – ФГОС) по специальности среднего профессионального образования (далее – СПО)

09.02.07 «Информационные системы и программирование»

Организация разработчик: Государственное бюджетное профессиональное образовательное учреждение Московской области «Воскресенский колледж»

Разработчик:

Комиссаров С.А., преподаватель компьютерных дисциплин

Рабочая программа профессионального модуля рассмотрена на заседании предметной (цикловой) комиссией компьютерных дисциплин

«___»_____2020г.

Председатель цикловой комиссии _____/Рязанцева О.В./

Утверждена зам директора по УР _____/Куприна Н.Л./

«___»_____2020 г.

ВВЕДЕНИЕ

Данные методические указания для проведения лабораторных работ по ПМ.02 «Осуществление интеграции программных модулей» предназначены для реализации ФГОС СПО по специальности 09.02.07 «Информационные системы и программирование» с целью закрепления теоретических знаний и практических умений.

В сборнике содержатся методические указания по выполнению 26 лабораторных работ по МДК.02.01 «Технология разработки программного обеспечения».

При выполнении лабораторных работ студент должен

иметь практический опыт:

- Разрабатывать и оформлять требования к программным модулям по предложенной документации.
- Разрабатывать тестовые наборы (пакеты) для программного модуля.
- Разрабатывать тестовые сценарии программного средства.
- Инспектировать разработанные программные модули на предмет соответствия стандартам кодирования.
- Интегрировать модули в программное обеспечение.
- Отлаживать программные модули.
- Инспектировать разработанные программные модули на предмет соответствия стандартам кодирования.;

уметь:

- Анализировать проектную и техническую документацию.
- Использовать специализированные графические средства построения и анализа архитектуры программных продуктов.
- Организовывать заданную интеграцию модулей в программные средства на базе имеющейся архитектуры и автоматизации бизнес-процессов.
- Определять источники и приемники данных.
- Проводить сравнительный анализ. Выполнять отладку, используя методы и инструменты условной компиляции (классы Debug и Trace).
- Оценивать размер минимального набора тестов.
- Разрабатывать тестовые пакеты и тестовые сценарии.
- Выявлять ошибки в системных компонентах на основе спецификаций.
- Использовать выбранную систему контроля версий.
- Использовать методы для получения кода с заданной функциональностью и степенью качества.
- Использовать различные транспортные протоколы и стандарты форматирования сообщений.
- Выполнять тестирование интеграции.
- Организовывать постобработку данных.
- Создавать классы-исключения на основе базовых классов.
- Выполнять ручное и автоматизированное тестирование программного модуля.
- Использовать приемы работы в системах контроля версий.
- Использовать инструментальные средства отладки программных продуктов.
- Использовать приемы работы в системах контроля версий.
- Выполнять отладку, используя методы и инструменты условной компиляции.
- Анализировать проектную и техническую документацию;

знать:

- Модели процесса разработки программного обеспечения.

- Основные принципы процесса разработки программного обеспечения.
- Основные подходы к интегрированию программных модулей.
- Виды и варианты интеграционных решений.
- Современные технологии и инструменты интеграции.
- Основные протоколы доступа к данным.
- Методы и способы идентификации сбоев и ошибок при интеграции приложений.
- Методы отладочных классов.
- Стандарты качества программной документации.
- Основы организации инспектирования и верификации.
- Встроенные и основные специализированные инструменты анализа качества программных продуктов.
- Графические средства проектирования архитектуры программных продуктов.
- Методы организации работы в команде разработчиков.
- Основы верификации программного обеспечения.
- Основные методы отладки.
- Методы и схемы обработки исключительных ситуаций.
- Основные методы и виды тестирования программных продуктов.
- Приемы работы с инструментальными средствами тестирования и отладки.
- Основные принципы процесса разработки программного обеспечения.
- Основные подходы к интегрированию программных модулей.
- Основы верификации и аттестации программного обеспечения;

Каждая лабораторная работа имеет следующую структуру: тема, цели, краткие теоретические сведения, порядок проведения работы, требования к составлению отчета.

После выполнения лабораторной работы студент должен представить отчет о проделанной работе. Оценку по практической работе студент получает, если студентом работа выполнена в полном объеме, студент может пояснить выполнение любого этапа работы, отчет выполнен в соответствии с требованиями к выполнению работы, студент отвечает на контрольные вопросы на удовлетворительную оценку и выше.

Зачет по выполнению лабораторных работ студент получает при условии выполнения всех предусмотренных программой лабораторных работ с отчетами по всем работам.

Отчет к лабораторной работе должен содержать (в зависимости от задания):

- Тему работы
- Задание для выполнения, включая индивидуальное задание
- Описание ключевых решений
- Описание выбранного ПО или оборудования
- Описание диаграмм
- Построенные диаграммы и схемы
- Выводы

Перечень лабораторных работ

1. Работа с классами. Разработка методов класса. Перегрузка методов.
2. Определение операций в классе. Создание наследованных классов
3. Работа с объектами через интерфейсы.
4. Использование стандартных интерфейсов.
5. Работа с типом данных структура.
6. Коллекции. Параметризованные классы.
7. Использование регулярных выражений
8. Операции со списками

Лабораторная работа №1.
Проведение предпроектных исследований.

Теоретическая часть.

Что бывает без предпроектного обследования?

Основные вопросы, на которые отвечает обследование

Как говорится, надо понять, ЧТО, ГДЕ, КОГДА. А именно:

С какой целью ведется разработка, какую выгоду извлечет заказчик.
Какова предлагаемая бизнес-схема, процесс, который будет автоматизирован с помощью создаваемой системы.
Каковы основные пользовательские функции системы.

Зачем нужно писать, почему недостаточно обсудить и проговорить?

Составление документа позволяет сформулировать мысль на совершенно ином качественном уровне, чем при устном обсуждении. В разговоре неохваченными остаются многие детали, часть информации забывается и позже упускается из виду. А бумага сохраняет все мысли.

Да, составление документов — дело кропотливое и иногда неприятное, но оно того стоит. Мысль ценна только тогда, когда она сформирована, а сформирована она тогда, когда сформулирована на бумаге.

Что должно содержать в себе предпроектное обследование?

Обычно под предпроектным обследованием имеют в виду изучение бизнес-процессов предприятия. Об этом написано много статей и книг. Но к сожалению, простого изложения процессов недостаточно.

Результатом исследования может быть целый пакет документов (часть из них приведена в конце статьи). Центральным (и, к сожалению, часто единственным) документом у меня обычно является документ «Концепция системы». Этот документ мы и обсудим в настоящей статьей.

Разрабатывая собственную структуру Концепции, я взял за основу отчет, подготавливаемый согласно ГОСТ 34 на стадии «Формирование требований к АС»

(см. стандарт РД 50-34.698-90 «Методические указания. Информационная технология. Комплекс стандартов и руководящих документов на автоматизированные системы. Автоматизированные системы. Требования к содержанию документов»). Но при этом внес свои дополнения.

«Концепция системы» может содержать 2, а иногда и 30 страниц. Все зависит от постановки задачи. «Концепция», как правило, согласовывается с высшим руководством заказчика, и только на основании этого можно разрабатывать Техническое задание.

Цель создания (модернизации) системы

Под целью создания понимаются именно бизнес-цели. «Автоматизировать» — это не цель. Добавить функцию — тоже не цель. И «оптимизировать» — не цель. Например, сидит сотрудник и пару часов в день он может поспать прямо на рабочем месте (реальный случай, кстати). И кто-то просит автоматизировать его деятельность. Зачем? Чтобы он спал четыре часа?

За несколько лет анализа десятков проектов удалось определить только пять возможных целей создания (модернизации) системы:

Организуется новый бизнес (например, онлайн-система заказов). Понятно, что, если бизнес планируется осуществлять через Интернет, без разработки не обойтись.
Снижение операционных расходов. Классический случай, когда в результате автоматизации сокращается персонал или удается с помощью более качественного планирования сделать больше с меньшими затратами.
Повышение качества внутренних процессов. Также классический случай. Например, если при поиске новых клиентов менеджеры постоянно забывают кому-то позвонить, теряют информацию о лиде, то имеет смысл внедрить CRM.
Снижение рисков при зависимости от ключевых сотрудников (этаких «золотых гвоздей»). Бывает, что из-за низкого уровня автоматизации и запутанности процессов ряд операций могут выполнить 1-2 сотрудника, увольнение (или болезнь) которых может поставить крест на всем бизнесе. А найти и научить новых займет не один месяц.
Выполнение внешних требований. Например, появился новый закон, или имеется требование контрагента, что у вас должен быть внедрен электронный документооборот или контроль за работой мобильных сотрудников.

Понятно, что цель желательно сделать осязаемой. Если мы хотим снизить расходы, то насколько и за счет чего. Если организуем новый бизнес, то надо понимать хотя бы примерный объем операций и количество операторов. Если повышаем качество процессов, следует очертить круг проблем и предложить решение.

Идея системы

В случае, если документ «Концепция» получается достаточно объемным, имеет смысл вначале кратко изложить самую суть системы, ее идею. Например, вы хотите создать какую-либо специализированную социальную сеть (ходите по музеям и делитесь впечатлениями). Я бы вначале описал потребность в общении между посетителями, а затем кратко — суть: разрабатывается мобильное

приложение, в котором пользователь может написать свои впечатления от того или иного экспоната.

Сравнение старого и нового

Самым эффективным способом понять суть создаваемой системы — идти как бы от противного.

Для этого необходимо:

кратко описать существующие процессы;
указать на их недостатки;
предложить новую схему, устраняющую описанные недостатки.

Цель данного раздела — именно обосновать необходимость внедрения новой схемы. Подробное описание бизнес-процессов лучше вынести в отдельный документ. Здесь мы концентрируемся на недостатках и предложениях.

На чем собираемся зарабатывать

Если разрабатывается приложение, с помощью которого планируется зарабатывать деньги, то обязательно нужно определить методы заработка: размещение рекламы, платная подписка, платные услуги, взимаемый процент и т.д. Выбранный способ (или способы) может сильно повлиять на разрабатываемую функциональность.

Заинтересованность сторон

Если для функционирования создаваемой системы необходимо участие других организаций, то обязательно нужно решить, как их привлечь к работе, заинтересовать. Иными словами, сначала выстраиваем всю бизнес-цепочку, потом уже все остальное.

Описание автоматизируемых процессов

Цель данного раздела — дать общее, но полное представление о процессе. Например, вы разрабатываете интернет-магазин. Очевидно, что нужен каталог, корзина, интеграция с банком-эквайером и доставка. Но вот вопросы возврата, отказа при доставке, отказа поставщика, неожиданного отсутствия товара на складе могут ускользнуть от вашего внимания. Лучше продумать все возможные варианты заранее и решить, что из этого будет автоматизироваться, а какие случаи происходят так редко, что лучше их «разгребать» в ручном режиме.

Для описания не обязательно приводить схемы. В общем случае обычный текстовый сценарий намного более полно раскрывает сущность действий.

Юридическое обеспечение

Нередко после создания системы оказывается, что в использующие приложение люди или организации нарушают закон. Поэтому вначале надо найти юридически чистую схему, а затем уже вырабатывать технические решения.

Перечень функций

Документ «Концепция» — это не Техническое задание, поэтому описываются бизнес-функции, верхний уровень. Нет никакого смысла на данном этапе говорить об авторизации и работе с профилем пользователя. Но дать общее представление о функциональности надо обязательно.

Требования к безопасности

Если вы разрабатываете финансовую систему или систему, содержащую строго конфиденциальные данные, то необходимо привести перечень стандартов безопасности. Например, требования к шифрованию хранимых или передаваемых данных. Не забывайте и о все ужесточающихся требованиях к обработке и хранению персональных данных.

Выбор варианта реализации системы

Иногда в зависимости от потребностей необходимо определить вид приложения (веб-приложение, нативное), платформу (Windows, Linux), общую архитектуру (один сервер или несколько кластеров), взять ли типовую систему и доработать или вести разработку с нуля. Для этого необходимо сравнить предлагаемые варианты и выбрать наиболее подходящий.

Другие документы предпроектного исследования

Как мы уже говорили выше, результатом хорошего, серьезного предпроектного исследования, проводимого не одну неделю целой командой, является целый пакет документов. Вот некоторые из них:

Концепция системы (документ, который мы обсуждали в настоящей статье).
Маркетинговое исследование.
Технико-экономическое обоснование.
План проекта, включая расчет трудоемкости и ресурсный план.
План маркетинговых мероприятий.
Смета проекта.
План возврата инвестиций.
Предварительное штатное расписание.
Архитектура системы.
Концепция безопасности (в случае большого объема описания меры безопасности можно вынести в отдельный документ.
Презентации для заказчика, потенциальных инвесторов и потенциальных клиентов.

Заключение

Главное, чтобы при прочтении концепции сложилось полное понимание, как это должно работать. А в остальном два документа с результатами исследования могут никак не быть похожими друг на друга. Соответственно и перечень разделов в вашем документе может сильно отличаться от приведенного выше.

Практическое задание.

Варианты предметных областей: 1) небольшая сеть автомобильных заправочных станций; 2) автоматизация работы городской сети мини-ресторанов быстрого питания; 3) автоматизация работы городских библиотек; 4) автоматизация работы ЖКХ в пределах одного муниципального района; 5) автоматизация производства и учета мебельной фабрики; 6) создание интернет-провайдера; 7) автоматизация выездной розничной торговли; 8) спортивно-массовые мероприятия районного масштаба; 9) создание охранной системы промышленного предприятия.

Практическое задание.

Получив и записав свою предметную область, выполните следующие действия или ответьте на следующие вопросы:

1. Где и сколько объектов автоматизации находится в вашем распоряжении? Запишите соответствующую информацию.
2. Составьте список документов, которые у вас должны появиться после предпроектного исследования.

Подробно опишите цель создания системы.

Лабораторная работа №2.
Построение организационно-
функциональной структуры компании.

Теоретическая часть.

Что такое организационная структура предприятия?
Говоря об организационной структуре, мы имеем в виду концептуальную схему, вокруг которой организуется группа людей, основу, на которой держатся все функции. Организационная структура предприятия — это, по сути, руководство для пользования, которое объясняет, как организация выстроена и как она работает. Если говорить конкретнее, то организационная структура описывает, как в компании принимаются решения и кто является ее лидером.
Почему необходимо разрабатывать организационную структуру предприятия?

- Организационная структура дает четкое понимание того, в каком направлении движется компания. Ясная структура — это инструмент, с помощью которого можно придерживаться порядка в принятии решений и преодолевать различные разногласия.
- Организационная структура связывает участников. Благодаря ей люди, присоединяющиеся к группе, имеют отличительные черты. В то же время и сама группа обладает определенными особенностями.
- Организационная структура формируется неизбежно. Любая организация по определению подразумевает какую-то структуру.

Элементы организационной структуры
Организационная структура любой организации будет зависеть от того, кто является ее участниками, какие задачи она решает и как далеко организация зашла в своем развитии.

Независимо от того, какую организационную структуру вы выбираете, три элемента всегда будут присутствовать в ней.

- Управление

Конкретный человек или группа людей, которые принимают решения в организации.

- Правила, по которым работает организация

Многие из этих правил могут быть заявлены явно, в то время как другие могут быть скрытыми, но при этом не менее обязательными для исполнения.

- Распределение труда

Распределение труда может быть формальным или неформальным, временным или постоянным, но в каждой организации непременно будет определенный тип распределения труда.

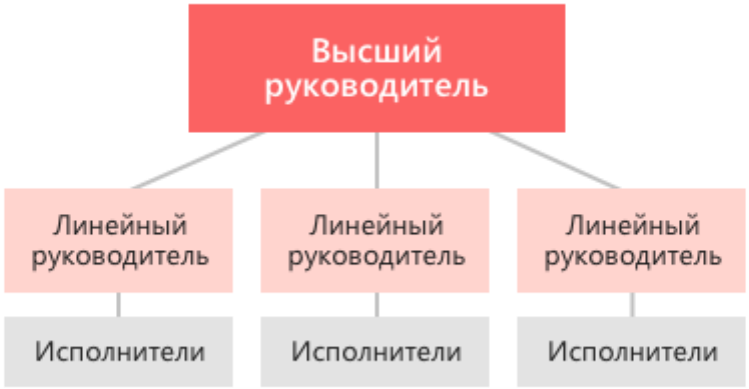
Традиционные организационные структуры

Эти структуры основаны на функциональном подразделении и отделах. Они характеризуются тем, что на верхнем уровне сосредоточены полномочия стратегических и оперативных задач.

Существует несколько типов традиционных структур.

- Линейная организационная структура

Самая простая структура из всех существующих. Характеризуется наличием определенной цепи инстанций. Решения спускаются сверху вниз. Этот вид структуры подходит для маленьких организаций вроде небольших бухгалтерских фирм и адвокатских контор. Линейная структура позволяет легко принимать решения.



Преимущества:

- Самый простой вид организационной структуры.
- В результате жесткого управления формируется жесткая дисциплина.
- Быстрые решения приводят к быстрым и эффективным действиям.
- В структурах власти и ответственности существует ясность.
- Поскольку контроль лежит на одном начальнике, в ряде случаев он может проявлять гибкость.
- Есть хорошие перспективы карьерного роста у людей, которые выполняют работу качественно.

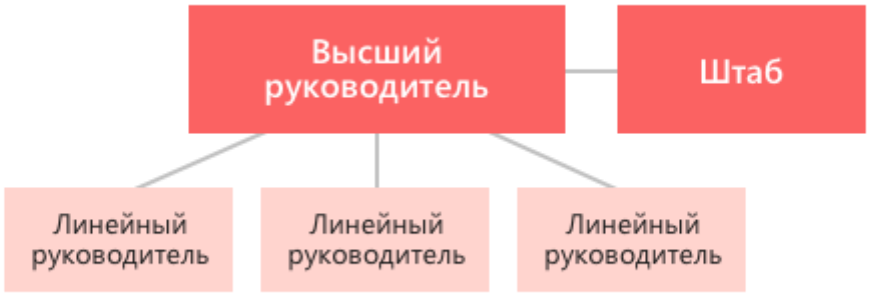
Недостатки:

- Есть возможности оказывать влияние на начальника отдела.
- Постоянная проблема — отсутствие специализации.
- Начальник отдела может быть перегружен работой.
- Коммуникации осуществляются только сверху вниз.
- Начальник, обладающий властью, может неправильно использовать ее для своей выгоды.

- Решения принимаются одним человеком.

Линейно-штабная организация

Такая структура характеризуется наличием линейных руководителей и подразделений, которые по факту не имеют права принятия решений. Главная их задача — оказывать помощь линейному менеджеру в выполнении отдельных функций управления. Процесс принятия решений в такой структуре медленнее.



Преимущества:

- Позволяет сотрудникам быстро выполнять задачи.
- Помогает сотрудникам брать на себя ответственные функции и специализироваться на конкретных функциях.
- Помогает линейным руководителям сконцентрироваться на определенных задачах.
- При организационных изменениях минимален риск возникновения сопротивления.
- Сотрудники чувствуют, что их вклад оценен.

Недостатки:

- Среди сотрудников может возникать путаница.
- У сотрудников недостаточно знаний, чтобы ориентироваться на результат.
- Слишком много уровней иерархии.
- Сотрудники могут расходиться во мнениях, что замедляет работу.
- Более дорогостоящая структура, чем простая линейная организация, из-за наличия начальников подразделений.
- Решения могут приниматься слишком долго.

Функциональная структура

Этот вид организационной структуры классифицирует людей согласно функции, которую они выполняют в профессиональной жизни.

Преимущества:

- Высокая степень специализации.
- Ясный порядок подчиненности.
- Четкое понимание ответственности.

- Высокая эффективность и скорость.
- Отсутствие необходимости в дублировании работы.
- Все функции одинаково важны.

Недостатки:

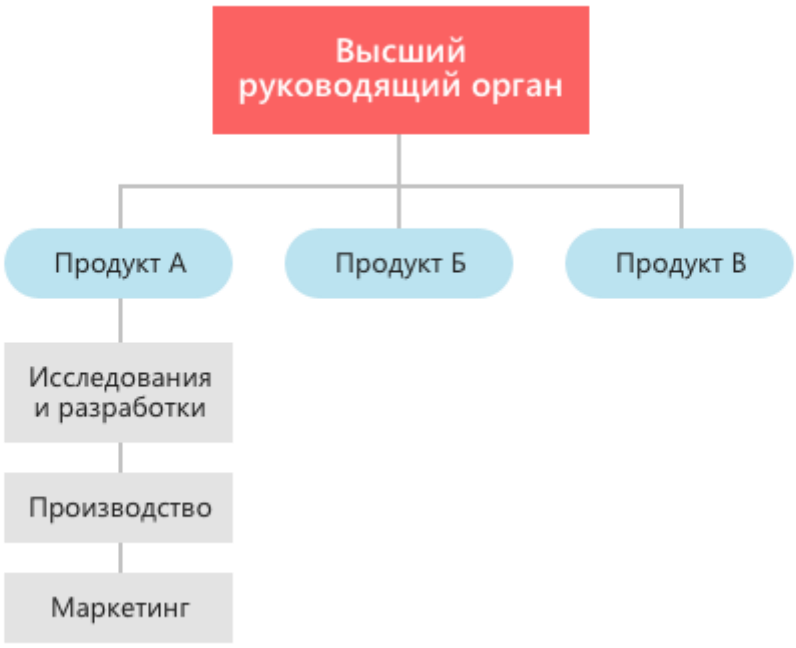
- Коммуникация сталкивается с несколькими барьерами.
- В центре внимания находятся люди, а не организация.
- Решения, принятые единственным человеком, могут не всегда идти на пользу организации.
- По мере роста компании становится труднее осуществлять контроль над действиями внутри нее.
- Отсутствие командной работы между различными отделами или единицами.
- Поскольку все функции отделены, сотрудники могут не знать о том, что творится у коллег.

Дивизиональная структура

Сюда относятся виды структур, которые основаны на различных подразделениях в организации. Они группируют сотрудников на основе продуктов, рынков и географического положения.

- Продуктовая (товарная) структура

Такая структура основана на организации сотрудников и работы вокруг различных продуктов. Если компания производит три различных продукта, то у нее будут три различных подразделения для этих продуктов. Этот тип структуры лучше всего подходит для розничных магазинов с множеством продуктов.



Преимущества:

- Структурные единицы, которые не работают, можно легко закрыть.
- Каждая единица может управляться как отдельное структурное подразделение.
- Быстрое и легкое принятие решений.
- Большая независимость у лиц, принимающих решения.
- Отдельные продукты привлекают отдельное внимание в зависимости от проблем, которые возникают.
- Организация характеризуется высокой производительностью и эффективностью.

Недостатки:

- Поскольку каждая структурная единица работает самостоятельно, организационные цели не могут быть достигнуты.
- Нездоровая конкуренция среди внутренних подразделений.
- Большое количество организационных уровней препятствует развитию бизнеса.
- Все единицы не могут быть равнозначными.
- Маркетинг отдельных продуктов может сильно отличаться по стоимости.

Рыночная структура

Сотрудники группируются исходя из того, на каком рынке работает компания.

У компании может быть пять различных рынков, согласно этой структуре каждый из них будет отдельным подразделением.

Преимущества:

- Сотрудники могут общаться с клиентами на местном языке.
- Они доступны клиентам.
- Проблемы на конкретном рынке могут решаться изолированно.
- Поскольку люди ответственны за конкретный рынок, задачи выполняются вовремя.
- Сотрудники специализируются на работе на конкретном рынке.
- Могут выводиться новые продукты для специализированных рынков.

Недостатки:

- Может возникнуть острая конкуренция среди сотрудников.
- Принятие решений может вызывать конфликты.
- Трудно определить производительность и эффективность.
- Все рынки могут не рассматриваться как равные.
- Может отсутствовать связь между начальниками и сотрудниками.
- Сотрудники могут неправильно использовать свои полномочия.
- Географическая структура

У крупных организаций есть офисы в различных местах. Организационная структура в этом случае следует за зональной структурой.

Преимущества:

- Хорошая коммуникация среди сотрудников в том же самом местоположении.
- Местные работники лучше знакомы с местной деловой средой и могут приспосабливаться к географическим и культурным особенностям.
- Клиенты чувствуют лучшую связь с местными менеджерами, которые могут говорить на их языке.
- Отчеты по работе отдельных рынков.
- Решения принимаются взвешенно.
- Могут вводиться новые продукты или модификации продуктов, удовлетворяющие потребности определенной области.

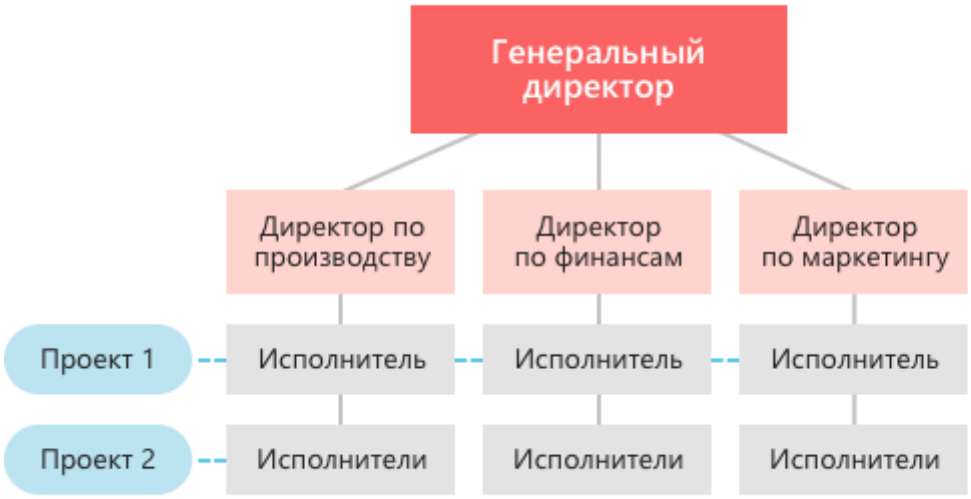
Недостатки:

- Может возникать нездоровая конкуренция среди различных географических зон.
- Этика компании и ее принципы могут отличаться от региона к региону.
- Отслеживание работы и прибыли каждой области может отнимать много времени.
- Возможна плохая коммуникация среди сотрудников в различных регионах.
- Взаимодействие между сотрудниками различных регионов может не сложиться.

Матричная структура

Это комбинация продуктовой и функциональной структур. Она объединяет преимущества обеих структур для большей эффективности. Эта структура самая сложная

из существующих. Отличительная особенность матричной структуры — подчинение сотрудников двум или более руководителям одного уровня.



Существует функциональная матрица. В этом типе матричной структуры менеджеры по проекту следят за функциональными аспектами проекта. Однако они обладают очень ограниченной властью, фактически управляет ресурсами и проектом руководитель функционального подразделения.

Преимущества:

- Сотрудники не работают на временной работе.
- Руководитель функционального подразделения управляет проектом.
- Руководитель функционального подразделения несет ответственность в случае, если что-либо идет не так, как надо.
- Чем больше менеджер по проекту общается с сотрудниками, тем лучше результаты.
- Менеджер по проекту может реально повлиять на ситуацию, не будучи под контролем.
- Принятие решений сосредоточено в руках руководителя функционального подразделения.

Недостатки:

- Менеджер по проекту может столкнуться с апатией со стороны сотрудников.
- Менеджер по проекту не имеет полной власти.
- Будучи не контролируемыми, сотрудники могут показывать меньшую производительность всего подразделения.
- Менеджер по проекту обладает слабой властью, которая не позволяет ему контролировать сотрудников.
- Менеджер по проекту не имеет никакого контроля над управлением рабочей нагрузкой и определением приоритетов в задачах.
- Менеджер по проекту не может дать отчет о работе.

Есть еще проектная матрица, когда прежде всего ответственен за работу менеджер по проекту, в то время как руководитель функционального подразделения может давать методические консультации и распределять ресурсы.

Практическое задание.

Варианты предметных областей: 1) небольшая сеть автомобильных заправочных станций; 2) автоматизация работы городской сети мини-ресторанов быстрого питания; 3) автоматизация работы городских библиотек; 4) автоматизация работы ЖКХ в пределах одного муниципального района; 5) автоматизация производства и учета мебельной фабрики; 6) создание интернет-провайдера; 7) автоматизация выездной розничной торговли; 8) спортивно-массовые мероприятия районного масштаба; 9) создание охранной системы промышленного предприятия.

Используя ранее выбранную предметную область, выполните следующие действия или ответьте на следующие вопросы:

3. Исследуйте модели организации внутренних структур. Используйте в качестве основы реально существующие бизнес-структуры.
4. Изучите связи внутри подразделениями и сотрудниками компании.
5. Постройте организационную модель вашей компании, учитывая все подразделения, в виде иерархической структуры, с использованием программных средств (Microsoft Visio или Draw.io).

Контрольные вопросы.

1. Зачем нужно составление структурной схемы предприятия (организации)?
2. Какие недостатки в структуре могут проявиться в ходе составления схемы организации?
3. В чём заключается суть функциональной структуры организации?
4. Насколько целесообразно построение рыночной структуры предприятия с точки зрения разработки программного обеспечения?
5. В каких случаях предпочтительнее построение матричной структуры?
6. От чего зависит степень детализации структуры предприятия?
7. Когда целесообразно использовать дивизиональную организационную структуру?

Какая структура из теоретической части является наиболее легко воспроизводимой с помощью программного обеспечения?

Лабораторная работа №3.
Исследование информационных потоков компании.

Теоретическая часть.

ПРИМЕЧАНИЕ: РАБОТА ЯВЛЯЕТСЯ БОЛЬШОЙ ПО ОБЪЁМУ И НЕ МОЖЕТ БЫТЬ ПРЕДСТАВЛЕНА В ДАННОМ ДОКУМЕНТЕ

Варианты предметных областей: 1) небольшая сеть автомобильных заправочных станций; 2) автоматизация работы городской сети мини-ресторанов быстрого питания; 3) автоматизация работы городских библиотек; 4) автоматизация работы ЖКХ в пределах одного муниципального района; 5) автоматизация производства и учета мебельной фабрики; 6) создание интернет-провайдера; 7) автоматизация выездной розничной торговли; 8) спортивно-массовые мероприятия районного масштаба; 9) создание охранной системы промышленного предприятия.

Практическое задание.

Используя ранее выбранную предметную область, выполните следующие действия или ответьте на следующие вопросы:

6. Выберите метод, который считаете приемлемым для выполнения исследования информационных потоков.
7. Опираясь на расчёт структуры предприятия, выполненный вами в предыдущих работах, постройте модель информационных потоков.

Контрольные вопросы.

1. Зачем нужна модель информационных потоков компании?
2. В чём преимущество методики построения модели с помощью графов?
3. В чём заключаются недостатки матричного метода?
4. Какой метод представления информационных потоков выглядит наиболее репрезентативным?
5. В каких случаях построение модели информационных потоков оказывается нецелесообразным?
6. С помощью какого метода наиболее просто перенести модель в компьютерную программу для управления процессом производства или оказания услуг?
7. Приведите примеры программного обеспечения, которые используют в своей модели информационных связей методы реквизитов.
8. Какие методы построения информационных моделей уместны для небольших организаций?
9. В чём преимущество и недостатки использования транспортной модели?
10. В чём суть семантического анализа при построении модели информационных потоков?
11. Какие элементы модели информационных потоков вы можете назвать?

Чем отличаются модели информационных потоков для разных организаций, есть ли принципиальные различия?

Лабораторная работа №4.
Составление статического описания компании.

Теоретическая часть.

Статическое описание компании заключается в построении организационно-функциональной модели компании. Статическое описание компании включает в себя описание бизнес-потенциала, функционала (классификатора бизнес-процессов) и зоны ответственности менеджмента компании.

Бизнес-потенциал компании - набор видов коммерческой деятельности (бизнесов), направленный на удовлетворение потребностей конкретных сегментов рынка. Специфики каналов сбыта формируют первоначальное представление об организационной структуре (определяются центры коммерческой ответственности). Возникает понимание основных ресурсов, необходимых для воспроизводства товарной номенклатуры.

Функционал организации отражает верхний уровень деления бизнес-процессов. В классификаторе бизнес-процессов (корневой модели) обычно выделяют четыре базовых раздела:

1. Основные бизнес-процессы (непосредственно ориентированы на производство продукции, представляющие ценность для клиента и обеспечивающие получение дохода для организации).
2. Обеспечивающие бизнес-процессы (бизнес-процессы, которые предназначены для обеспечения выполнения основных процессов. Фактически обеспечивающие бизнес-процессы снабжают ресурсами всю деятельность организации).
3. Бизнес-процессы управления (бизнес-процессы, охватывающие весь комплекс функций управления на уровне текущих действий и бизнес-системы в целом).
4. Бизнес-процессы развития (процессы совершенствования, освоения новых направлений и технологий, а также инновации).
Для каждого из выделенных обеспечивающих процессов следует определить, какой основной или управленческий процесс является потребителем этих "внутренних" услуг.
В результате установления функционала компании формируется **корневая модель** бизнес-процессов и создается **единая терминология** описания функций предприятия.
Для формирования основных функций менеджмента (процессов управления) компании сначала разрабатываются и утверждаются два базовых классификатора - "Компоненты менеджмента" (перечень используемых в организации контуров управления, например, управление закупками, персоналом, производством и др.) и "Этапы управленческого цикла" (технологическая цепочка операций, последовательно реализуемых менеджерами при организации работ в любом контуре управления). Далее аналогично, с помощью матрицы проекций, формируется список основных функций менеджмента.
Формирование зон ответственности за функционал компании выполняется с помощью матрицы организационных проекций.
Матрица организационных проекций представляет собой таблицу, в строках которой расположен список исполнительных звеньев, в столбцах - список функций, выполняемых в компании. Для каждой функции определяется исполнительное звено, отвечающее за эту функцию.
Заполнение такой таблицы позволяет по каждой функции найти исполняющие ее подразделения или сотрудника. Анализ заполненной таблицы позволяет увидеть "пробелы" как в исполнении функций, так и в загруженности сотрудников, а также рационально перераспределить все задачи между исполнителями и закрепить как систему в документе "Положение об организационной структуре".
В результате формирования функционала компании и зон ответственности строится организационно-функциональная структура компании (рис. 1).

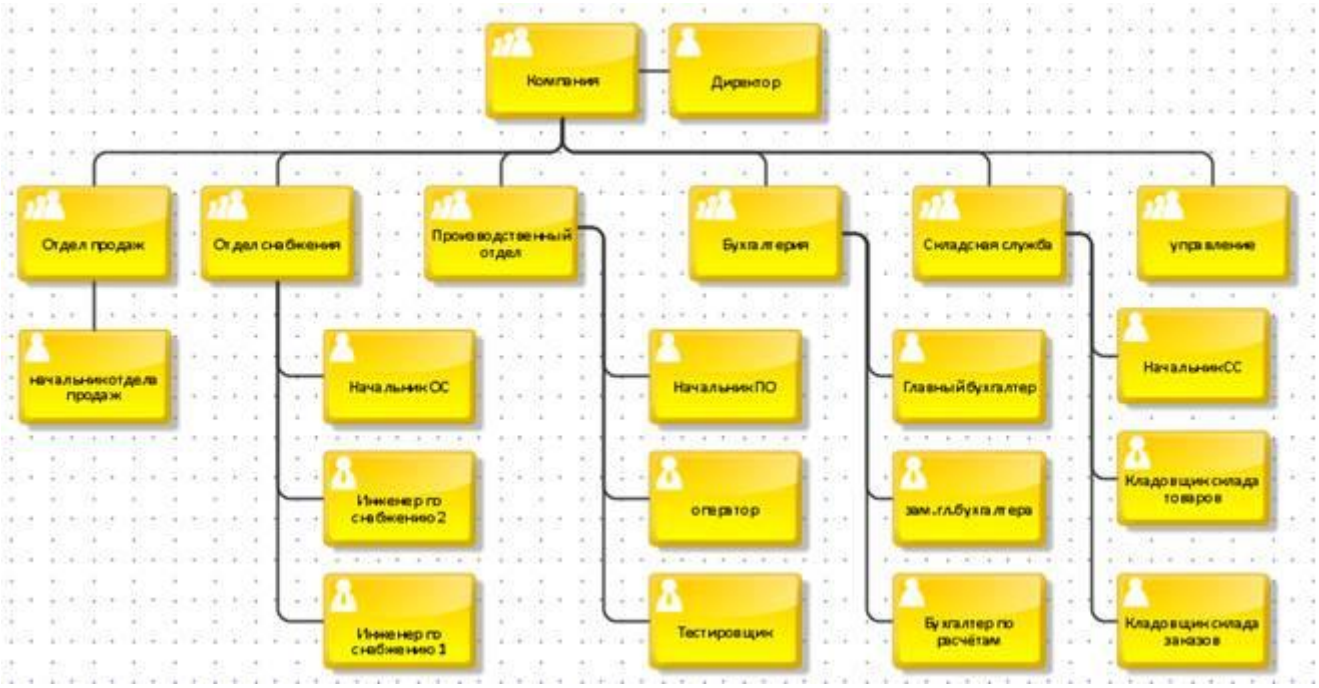


Рис. 1

Результаты выделения бизнес-процессов оформляются в виде табл. 3.

Таблица 3

Название БП	Вход(ы)	Поставщики	Выход(ы)	Потребители	Механизмы	Владелец
Управление деятельностью и развитием	Информация о макроокружении, информация по показателям БП	Окружающая среда	Решения по управлению	все бизнес процессы компании	Директор компании	Директор компании
…	…	…	…	…	…	…

Результаты выделения бизнес процессов организации отражаются в виде диаграммы декомпозиции контекстной модели организации, выполненной по методологии IDEF0 в стандарте SADT.

На основании статического описания компании путём обоснованного выбора бизнес-процессов формируется первичный перечень модулей информационной системы, например, модуль управления закупками, модуль управления продажами или услугами, кадры и др.

На этом этапе бизнес-моделирования формируется общепризнанный набор основополагающих внутрифирменных регламентов:

базовое Положение об организационно-функциональной структуре компании;

- пакет Положений об отдельных видах деятельности (финансовой, маркетинговой и т.д.);
- пакет Положений о структурных подразделениях (цехах, отделах, секторах, группах и т.п.);
- должностные инструкции.

Это вносит прозрачность в деятельность компании за счет четкого разграничения и документального закрепления зон ответственности менеджеров.

Матрица коммерческой ответственности закрепляет ответственность структурных подразделений за получение дохода в компании от реализации коммерческой деятельности.
Матрица функциональной ответственности закрепляет ответственность структурных звеньев (и отдельных специалистов) за выполнение бизнес-функций при реализации процессов коммерческой деятельности (закупка, производство, сбыт), а также функций менеджмента, связанных с управлением этими процессами (планирование, учет).

Практическое задание.

Варианты предметных областей: 1) небольшая сеть автомобильных заправочных станций; 2) автоматизация работы городской сети мини-ресторанов быстрого питания; 3) автоматизация работы городских библиотек; 4) автоматизация работы ЖКХ в пределах одного муниципального района; 5) автоматизация производства и учета мебельной фабрики; 6) создание интернет-провайдера; 7) автоматизация выездной розничной торговли; 8) спортивно-массовые мероприятия районного масштаба; 9) создание охранной системы промышленного предприятия.

Используя ранее выбранную предметную область, выполните следующие действия или ответьте на следующие вопросы:

1. Постройте статическую бизнес-модель компании.
2. Создайте матрицу бизнес-проекций.
3. Создайте матрицу коммерческой ответственности.
4. Создайте матрицу функциональной ответственности.

При необходимости используйте средства построения графиков и диаграмм.

Лабораторная работа №5.
Составление динамического описания компании.

Теоретическая часть.

Создание систем эффективного управления предприятиями является первостепенной задачей современного менеджмента. В настоящее время для большинства российских предприятий характерна функциональная иерархическая модель управления, обладающая рядом существенных недостатков. Функциональный подход характеризуется четкой специализацией труда, из-за чего отдельные группы сотрудников «не видят» конечных результатов работы всей компании, а значит, не могут быть заинтересованными в их достижении. Разногласия между работниками, несогласованность действий функциональных подразделений существенно снижают эффективность деятельности предприятия в целом.
На современном этапе развития управления общемировой тенденцией является применение процессного подхода к управлению деятельностью. Основу практического воплощения процессного подхода составляет формализация процессов предприятия и декомпозиция процессов до операций. Это позволяет разрабатывать оптимальные способы формирования управленческих данных и принятия управленческих решений, определяет уровни компетенции и разделяет уровни управления по менеджерам предприятия. Корме того, в ходе описания бизнес-процессов зачастую выявляются скрытые связи и взаимоотношения, использование которых в конечном итоге может

приводить к существенному повышению эффективности функционирования всего процесса.

При рассмотрении предприятия как логистической системы становится очевидным, что лишь при единой взаимосвязи и скоординированности бизнес - процессов можно говорить о качественном менеджменте. Поэтому для повышения уровня организации материальных потоков автором предлагается использовать положения процессного подхода. Применение процессного подхода в логистике, и в частности при организации материальных потоков, обусловлено его большим вниманием к описанию разделения задач и потоков информации между структурами, распределению ответственности и полномочий менеджеров, что позволяет целенаправленно улучшать отдельные звенья логистической цепи в сквозной единой системе информационных, материальных и финансовых потоков.

Далее рассмотрим процессную модель организации материальных потоков в производственно-сбытовых системах, которая иллюстрирует связи между всеми имеющимися процессами. Все Процессы разделяются на Основные и Вспомогательные. В качестве Основных в данной модели выделены процессы, связанные с товародвижением внутри предприятия. Вспомогательными Процессами модели выступают все действия по созданию нормальных условий для осуществления процесса движения материальных потоков. Общий вид модели представлен на рисунке 1.

Рисунок 1 – Процессная модель организации материальных потоков

Как видно из рисунка 1, в ходе удовлетворения потребительского спроса, т.е. выполнения поступивших заказов, на предприятиях осуществляется несколько видов операций:

- закупка сырья, материалов и комплектующих;
- операции технологической обработки, связанные с изменением форм, размеров, состояния, взаимного расположения деталей и узлов, составляющих изделие;
- сбыт готовой продукции потребителям.

При этом в процессе их осуществления требуется выполнить работы по погрузке-разгрузке, внутризаводскому перемещению материалов и продукции, их складированию и хранению. Таким образом, перечисленные процессы выполнения заказа дополняются транспортировкой и складированием (с сопутствующими им операциями), которые как бы пронизывают их на всем протяжении – от закупки материальных ресурсов до сбыта готовой продукции.

Основные Процессы для более эффективного управления в свою очередь можно детализировать на укрупненные задачи. Далее задачи можно расписать по элементарным операциям. Например, закупка состоит из определения размера потребности в материалах, формирования заявки на закупку, выбора поставщика, утверждения заявки на закупку, формирование заявки поставщику, получения счета, получения разрешения на оплату, оплаты по счету, доставки, передачи на склад.

Помимо Основных Процессов при организации материальных потоков необходимо решить ряд задач по обеспечению нормального его протекания. Они составляют совокупность Вспомогательных Процессов. В данной модели автором выделены следующие из них: ресурсное обеспечение, управление инфраструктурой, управление средствами труда, Обеспечение качества, управление документацией, управление персоналом, внутренний аудит. Не вдаваясь в детализацию, охарактеризуем предназначение каждого Вспомогательного процесса.

Ресурсное обеспечение призвано решать задачи получения и доставки в производственные подразделения точно в срок всего комплекса ресурсов, необходимых для эффективной работы по организации материальных потоков. Оно включает в себя: методическое, информационное, материальное, кадровое и техническое обеспечение.
Управление производственной инфраструктурой предполагает деятельность по решению задач ремонта и технического обслуживания оборудования, инструментального производства и обслуживания, организацию складских и транспортных работ, энергетическое обеспечение производства.
Управление орудиями труда включает задачи по формированию оптимальной структуры парка оборудования, его систематическому обновлению, улучшению загрузки оборудования и использованию его технических возможностей, повышению эффективности ремонта оборудования и его технического обслуживания.
Обеспечение качества состоит из комплекса задач по достижению высокого уровня качества протекания процессов на всех стадиях товародвижения за счет внедрения прогрессивных систем и методов контроля и улучшения организации труда работников.
Управление документацией включает процессы формирования информационной модели предприятия и его подразделений: разработка информационных потоков; отбор информации, необходимой для того или иного уровня управления и соответствующих подсистем; передача информации всем подразделениям предприятия.
Управление персоналом применительно к организации материальных потоков сконцентрировано на организации труда её участников и решении задач по подготовке и повышению квалификации кадров, а также разработке и поддержании в рабочем состоянии организационно-экономического механизма их взаимодействия.
Внутренний аудит – призван оперативно выявлять отклонения в процессе движения материальных потоков и осуществлять корректировочные действия по их устранению.
Заканчивая описание процессной модели организации материальных потоков, следует сказать, что реализация процесса, как Основного, так и Вспомогательного, должна рассматриваться как цепочка задач. Исполнение какой-либо задачи инициирует запуск следующей задачи. Контроль исполнения задач во времени обеспечивает контроль течения всех процессов деятельности предприятия.

Практическое задание.

Варианты предметных областей: 1) небольшая сеть автомобильных заправочных станций; 2) автоматизация работы городской сети мини-ресторанов быстрого питания; 3) автоматизация работы городских библиотек; 4) автоматизация работы ЖКХ в пределах одного муниципального района; 5) автоматизация производства и учета мебельной фабрики; 6) создание интернет-провайдера; 7) автоматизация выездной розничной торговли; 8) спортивно-массовые мероприятия районного масштаба; 9) создание охранной системы промышленного предприятия.

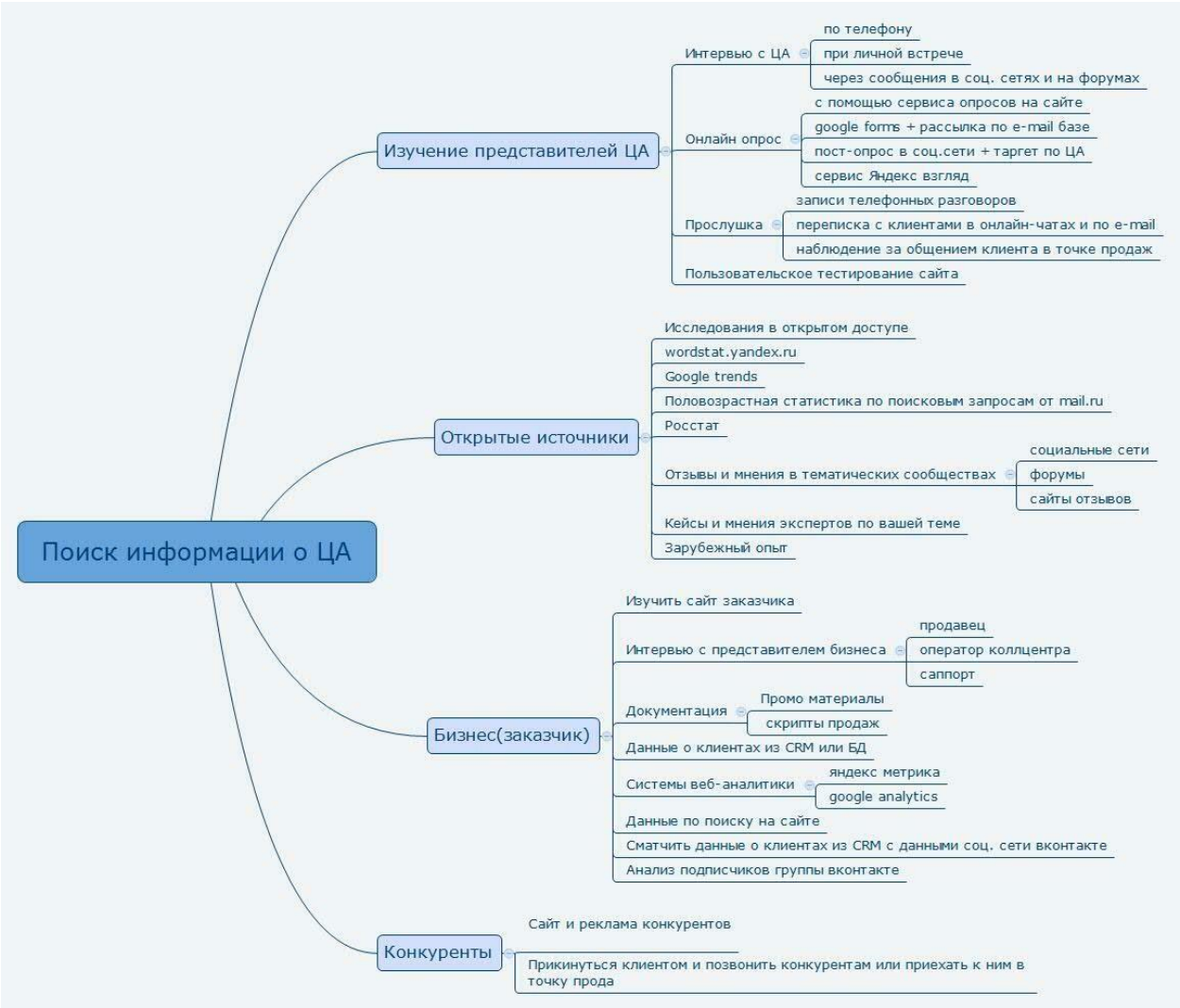
Используя ранее выбранную предметную область, выполните следующие действия или ответьте на следующие вопросы:

1. Постройте статическую бизнес-модель компании.
2. Создайте матрицу бизнес-проекций.
3. Создайте матрицу коммерческой ответственности.
4. Создайте матрицу функциональной ответственности.

При необходимости используйте средства построения графиков и диаграмм.

Лабораторная работа №9. Определение целевой аудитории программы.

Теоретический материал.



Целевая аудитория для программного обеспечения – это группа его основных заказчиков и потребителей. Она зависит не только от специфики ПО, но и от многих других факторов, например, текущей ситуации на рынке ПО, количества конкурентных предложений на рынке и т.п. Целевую аудиторию программного обеспечения необходимо продумывать заранее, ещё на стадии предпроектного анализа, и в обязательном порядке учитывать её предпочтения и пожелания при проектировании. Необходим тщательный анализ рынка программного обеспечения аналогичного назначения, необходимо понять, какие может преимущества дать наша разработка, в чём мы можем опередить конкурентов и какова наша ниша на этом рынке.

Изучение представителей ЦА

Это самое мощное направление по сбору информации о ЦА, т.к. здесь мы будем получать информацию непосредственно от людей, которые являются представителями ЦА

Открытые источники данных

Дополнительные материалы на тему:

https://www.insales.ru/blogs/university/celevaja-auditorija

https://compress.ru/article.aspx?id=18558

Практическое задание. Используя предметные области (см. список ниже), выполните следующие действия:

1. Опишите приблизительный портрет основного потребителя вашего программного продукта: пол, возраст, социальное положение, профессия и т.п.
2. Перечислите методы, с помощью которых вы планируете составлять портрет целевой аудитории вашего программного продукта.
3. Подробно опишите как минимум один из этих методов, включая необходимые для этого технические средства.
4. Проведите исследование ЦА с помощью доступной методики (например, Гугл-анализа) и сравните полученный результат с тем портретом основного потребителя, который вы составили изначально.

Список предметных областей:

Варианты предметных областей: 1) небольшая сеть автомобильных заправочных станций; 2) автоматизация работы городской сети мини-ресторанов быстрого питания; 3) автоматизация работы городских библиотек; 4) автоматизация работы ЖКХ в пределах одного муниципального района; 5) автоматизация производства и учета мебельной фабрики; 6) создание интернет-провайдера; 7) автоматизация выездной розничной торговли; 8) спортивно-массовые мероприятия районного масштаба; 9) создание охранной системы промышленного предприятия; 10) создание АРМ сотрудников юридической консультации; 11) создание системы учета технологических операций автосборочного завода; 12) создание системы учета продаж и заказов автосалона.

Список контрольных вопросов:

1. Что такое целевая аудитория ПО?
2. Что такое хорошо подобранная ЦА?
3. На каком этапе необходим анализ ЦА?
4. Какие методы определения ЦА достигаются без привлечения социальных методик?
5. Что такое сегментирование?
6. В чем заключается методика 5W?
7. Как проводится сегментация ЦА?
8. В чем проблема слишком широкой ЦА?
9. В чём проблема слишком узкоспециализированной ЦА?
10. Как исследование рынка ПО может помочь составить портрет клиента?
11. Чем packaged SW отличается от заказного ПО и в чём разница для потенциального клиента?

Зачем нужно позиционирование ПО?

Лабораторная работа №10. Выработка концепции ПП.

Теоретический материал.

Создание концепции продукта

Именно на этом этапе идея нового устройства принимает очертания коммерчески успешного продукта. Идет оценка конкурентов и составление сравнительной таблицы — карты технического уровня (КТУ). Проходит выбор оптимального сочетания функционально-технических характеристик продукта и определяется бизнес-модель всего проекта.

Технические эксперты разрабатывают технико-коммерческое предложение и концепт-дизайн продукта с описанием его преимуществ и функциональных особенностей. В это же время бизнес-аналитики создают бизнес-план. На этой стадии могут проводиться дополнительные маркетинговые исследования, поиск инвестиций и другие активности, основная цель которых — получить максимально приближенную к реальности «картинку» жизненного цикла продукта.

Концепция изделия может содержать следующую информацию:

- Функциональное описание устройства и описание решаемых задач
- Техническое описание устройства
- Описание эксплуатационных характеристик
- Предварительные эскизы и/или фотореалистичные изображения устройства
- Маркетинговый план с описанием:

1. Прямых и функциональных конкурентов устройства
2. Целевой аудитории
3. Рынков сбыта и зависимостей
4. Типовых сценариев использования
5. Предполагаемой бизнес-модели

Результаты этапа: технико-коммерческое предложение на разработку устройства, а также пакет документов, отражающий концепцию продукта (она может незначительно изменяться на последующих этапах).

Этапы разработки концепции

1. **Определение целей**. На основании анализа целей и задач организации, уже имеющихся информационных ресурсов и планов их развития, групп потенциальных пользователей, а также ресурсов, доступных для разработки и эксплуатации автоматизированной системы, можно сформулировать цели проекта. При этом целесообразно учесть изменения целей, задач и ресурсов на несколько лет вперед.

2. **Определение основных задач**. Для конкретной автоматизированной системы целесообразно указать (без детализации) конкретные задачи, которые должна решать эта система в течении следующих нескольких лет –- какую информацию собирать, в каком виде и как часто распространять.

3. **Определение допущений и рисков**. На данном этапе желательно приближенно оценить использованные при предварительной оценке проекта допущения, а также возможные риски во время разработки и эксплуатации автоматизированной системы. В качестве типового допущения обычно предполагается, что ресурсы на разработку и эксплуатацию автоматизированной системы будут выделяться своевременно и в нужном объеме. Примером возможного риска является случай, когда руководство предполагает разработать автоматизированную систему, взаимодействующую с интернетом, силами своих сотрудников, не имеющих такого опыта.

4. **Согласование результатов анализа**. Необходимо проводить в первую очередь с группами потенциальных пользователей. Целесообразно сразу предложить несколько вариантов автоматизированной системы, различающихся набором функций и затратами на разработку и эксплуатацию:

- базовый вариант,
- более дешевый, с меньшим набором функций,
- более дорогой, с расширенными возможностями.

Практическое задание. Используя предметные области (см. список ниже), а также прикрепленный к работе файл формата PDF с документом «Концепция создания» выполните следующие действия:

1. Разработайте основные пункты Концепции создания вашего программного продукта.
2. Определите состав подсистем вашего программного продукта: основные объекты, базы данных, библиотеки и т.п.
3. Опишите подробно те задачи, которые решает разработка концепции создания ПП.

Предложите несколько вариантов ПП – базовый, более дорогой и более дешёвый, и представьте эту информацию в виде таблицы.

Лабораторная работа №11-12. Составление технического задания.

Теоретический материал.

Что такое техническое задание на разработку программного обеспечения?

Большинство разработчиков предпочитают работать с техническим заданием на разработку программного обеспечения, так как этот документ обычно содержит следующее:

- Полное описание целей и функциональности программного обеспечения;
- Детали того, как программа будет работать с точки зрения скорости, времени отклика, доступности, мобильности, надёжности, скорости восстановления и т.д.;
- Варианты того, как пользователи будут использовать программное обеспечение;
- Определение того, как приложение будет взаимодействовать с оборудованием или другими программами;
- Нефункциональные требования (например: требования к обеспечению эффективности, стандарты качества, или проектные ограничения)

Нет стандартного метода написания ТЗ, но мы можем дать несколько советов:

Создайте схему

Если у вас ещё нет шаблона, их можно найти в Интернете. Используйте шаблон для создания плана документа. Измените его в соответствии с потребностями вашей организации.

Планы технических заданий различаются в зависимости от организации и её процессов. Некоторые из них могут быть простыми, другие являются более подробными и сложными.

Вот пример простого плана ТЗ на ПО:

1. Сфера применения
2. Обзор системы
3. Ссылки
4. Определения
5. Примеры использования
6. Функциональные требования
7. Нефункциональные требования

После создания плана можно писать спецификацию. Вот несколько советов:

Выберите для написания лучшего

Писатель должен иметь превосходные коммуникационные навыки. Цель спецификации в том, чтобы её мог понять каждый. Всё, что остается неясным или недопонятым, может привести к не особо приятным последствиям. Многие предполагают, что участие в процессе технического писателя помогает предотвратить непонимание. Есть писатели, более опытные, чем разработчики, с талантом вносить точность и ясность. Технические писатели знают, как собирать и обрабатывать нужную информацию; они также знают, как донести требования заказчика.

Сделайте информацию визуальной

Изображение может сэкономить 1000 слов. Включите визуальную информацию, например, таблицы и графики, чтобы лучше донести идеи.

Не документируйте слишком много

Старайтесь не включать в документ пункты, которые не нужно документировать. ТЗ может стать слишком длинным, поэтому избегайте лишней информации.

Создайте онлайн-версию ТЗ и постоянно обновляйте её

По мере выполнения задач или если произошли изменения в штате или процессах, ТЗ необходимо будет обновлять. По этой причине сохраняйте виртуальную версию – это поможет убедиться, что вся команда при любом изменении получит обновлённый документ.

Образец технического задания на разработку программы.

1. Введение

1.1. Наименование программы

1.2. Назначение и область применения программы

2. Требования к программе

2.1. Требования к функциональным характеристикам программы

2.2. Требования к надежности программы

2.2.1. Требования к обеспечению надежного функционирования программы

2.2.2. Время восстановления программы после отказа

2.2.3. Отказы программы из-за некоректных действий оператора

3. Условия эксплуатации программы 3.1. Климатические условия эксплуатации программы

3.2. Требования к квалификации и численности персонала

3.3. Требования к составу и параметрам технических средств

3.4. Требования к информационной совместимости

3.4.1. Требования к информационным структурам и методам решения

3.4.2. Требования к исходным кодам и языкам программирования

3.4.3. Требования к программным средствам, используемым программой

3.4.4. Требования к защите информации и программ

3.5. Специальные требования

4. Требования к программной документации

4.1. Предварительный состав программной документации

5. Технико-экономические показатели

5.1. Экономические преимущества разработки программы

6. Стадии и этапы разработки программы

6.1. Стадии разработки программы

6.2. Этапы разработки программы

6.3. Содержание работ по этапам

7. Порядок контроля и приемки

1. Введение

1.1. Наименование программы

Наименование программы: "Тестовая программа"

1.2. Назначение и область применения

Программа предназначена для...

2. Требования к программе

2.1. Требования к функциональным характеристикам

Программа должна обеспечивать возможность выполнения перечисленных ниже функций:

2.2. Требования к надежности

2.2.1 Требования к обеспечению надежного функционирования программы

Надежное (устойчивое) функционирование программы должно быть обеспечено выполнением Заказчиком совокупности организационно-технических мероприятий, перечень которых приведен ниже:

а) организацией бесперебойного питания технических средств;

б) использованием лицензионного программного обеспечения;

в) регулярным выполнением рекомендаций Министерства труда и социального развития РФ, изложенных в Постановлении от 23 июля 1998 г.

Об утверждении межотраслевых типовых норм времени на работы по сервисному обслуживанию ПЭВМ и оргтехники и сопровождению программных средств»;

г) регулярным выполнением требований ГОСТ 51188-98. Защита информации. Испытания программных средств на наличие компьютерных вирусов

2.2.2. Время восстановления после отказа

Время восстановления после отказа, вызванного сбоем электропитания технических средств (иными внешними факторами), не фатальным сбоем (не крахом) операционной системы,

не должно превышать 30-ти минут при условии соблюдения условий эксплуатации технических и программных средств.

Время восстановления после отказа, вызванного неисправностью технических средств, фатальным сбоем (крахом) операционной системы, не должно превышать времени, требуемого на устранение неисправностей технических средств и переустановки программных средств.

2.2.3. Отказы из-за некорректных действий оператора

Отказы программы возможны вследствие некорректных действий оператора (пользователя) при взаимодействии с операционной системой.

Во избежание возникновения отказов программы по указанной выше причине следует обеспечить работу конечного пользователя без предоставления ему административных привилегий

3. Условия эксплуатации

3.1. Климатические условия эксплуатации

Климатические условия эксплуатации, при которых должны обеспечиваться заданные характеристики, должны удовлетворять требованиям,

предъявляемым к техническим средствам в части условий их эксплуатации

3.2. Требования к квалификации и численности персонала

Минимальное количество персонала, требуемого для работы программы, должно составлять не менее 2 штатных единиц — системный администратор и конечный пользователь программы — оператор.

Системный администратор должен иметь высшее профильное образование и сертификаты компании-производителя операционной системы. В перечень задач, выполняемых системным администратором, должны входить:

а) задача поддержания работоспособности технических средств;

б) задачи установки (инсталляции) и поддержания работоспособности системных программных средств — операционной системы;

в) задача установки (инсталляции) программы.

г) задача создания резервных копий базы данных.

3.3. Требования к составу и параметрам технических средств

3.3.1. В состав технических средств должен входить IBM-совместимый персональный компьютер (ПЭВМ), выполняющий роль сервера, включающий в себя:

3.3.1.1. процессор Pentium-2.0Hz, не менее;

3.3.1.2. оперативную память объемом, 1Гигабайт, не менее;

3.3.1.3. оперативную память объемом, 1Гигабайт, не менее;

3.3.1.4. операционную систему Windows 2000 Server или Windows 2003;

3.3.1.5. операционную систему Windows 2000 Server или Windows 2003;

3.3.1.6. Microsoft SQL Server 2000

3.4. Требования к информационной и программной совместимости

3.4.1. Требования к информационным структурам и методам решения

База данных работает под управлением Microsoft SQL Server. Используется много поточный доступ к базе данных. Необходимо обеспечить одновременную работу с программой с той же базой данной модулей экспорта внешних данных.

3.4.2. Требования к исходным кодам и языкам программирования

Дополнительные требования не предъявляются

3.4.3. Требования к программным средствам, используемым программой

Системные программные средства, используемые программой, должны быть представлены лицензионной локализованной версией операционной системы Windows 2000 Server или Windows 2003 и Microsoft SQL Server 2000

3.4.4. Требования к защите информации и программ

Требования к защите информации и программ не предъявляются

3.5. Специальные требования

Специальные требования к данной программе не предьявляются

4. Требования к программной документации

4.1. Предварительный состав программной документации

Состав программной документации должен включать в себя:

4.1.1. техническое задание;

4.1.2. программу и методики испытаний;

4.1.3. руководство оператора;

5. Технико-экономические показатели

5.1. Экономические преимущества разработки

Ориентировочная экономическая эффективность не рассчитываются. Аналогия не проводится ввиду уникальности предъявляемых требований к разработке.

6. Стадии и этапы разработки

6.1. Стадии разработки

Разработка должна быть проведена в три стадии:

1. разработка технического задания;

2. рабочее проектирование;

3. внедрение.

6.2. Этапы разработки

На стадии разработки технического задания должен быть выполнен этап разработки, согласования и утверждения настоящего технического задания.

На стадии рабочего проектирования должны быть выполнены перечисленные ниже этапы работ:

1. разработка программы;

2. разработка программной документации;

3. испытания программы.

На стадии внедрения должен быть выполнен этап разработки подготовка и передача программы

На этапе разработки технического задания должны быть выполнены перечисленные ниже работы:

1. постановка задачи;
2. определение и уточнение требований к техническим средствам;
3. определение требований к программе;
4. определение стадий, этапов и сроков разработки программы и документации на неё;
5. согласование и утверждение технического задания.

На этапе разработки программы должна быть выполнена работа по программированию (кодированию) и отладке программы.

На этапе разработки программной документации должна быть выполнена разработка программных документов в соответствии с требованиями к составу документации.

На этапе испытаний программы должны быть выполнены перечисленные ниже виды работ:

1. разработка, согласование и утверждение и методики испытаний;
2. проведение приемо-сдаточных испытаний;
3. корректировка программы и программной документации по результатам испытаний.

На этапе подготовки и передачи программы должна быть выполнена работа по подготовке и передаче программы и программной документации в эксплуатацию на объектах Заказчика.

7. Порядок контроля и приемки

7.1. Виды испытаний

Приемо-сдаточные испытания должны проводиться на объекте Заказчика в оговоренные сроки.

Приемо-сдаточные испытания программы должны проводиться согласно разработанной Исполнителем и согласованной Заказчиком Программы и методик испытаний.

Ход проведения приемо-сдаточных испытаний Заказчик и Исполнитель документируют в Протоколе проведения испытаний

7.2. Общие требования к приемке работы

На основании Протокола проведения испытаний Исполнитель совместно с Заказчиком подписывает Акт приемки-сдачи программы в эксплуатацию.

Еще один пример технического задания.

Техническое задание на разработку «ПМК для автоматизации обработки и аппроксимации экспериментальных данных»

Основания для разработки

Основанием для разработки является тема индивидуального задания для дипломной работы «ПМК для автоматизации обработки и аппроксимации экспериментальных данных» и дисциплины ППС.

Спец часть: Разработка ПО для распознавания делительных сеток и аппроксимации.

Назначение разработки

Программный комплекс предназначен для обработки и аппроксимации экспериментальных данных и должен выполнять следующие функции:

Распознавать изображение;

Выполнять сглаживание изображения;

Выделять делительные узлы.

Требования к программному изделию

При реализации и использовании информационной системы должны быть учтены требования к функциональным характеристикам, надежности проекта, параметрам технических средств, информационной и программной совместимости.

Требования к функциональным характеристикам

Программный комплекс должен выполнять следующие функции:

Распознание изображения (разрешающая способность не менее 600×300 DPI);

Выделение делительных узлов (не более 1000 узлов) ;

Вычисление координат делительных узлов (с точностью 0,01 мм);

Формирование массива данных (не более 30 секунд);

Сглаживание изображения (2 и более метода для аппроксимации в зависимости кривые или поверхности);

Сохранение результата (более 3 форматов).

Требования к надежности

Программный продукт должен устойчиво функционировать и не приводить к сбоям операционной системы в аварийных ситуациях. В случае возникновения сбоя должны выдаваться корректные сообщения с указанием дальнейших действий. Для предупреждения возникновения ошибок необходимо наличие руководства по эксплуатации ПМК.

Программный продукт должен обеспечивать контроль входной и выходной информации на соответствие заданным форматам данных.

Программный продукт должен обеспечивать обработку ошибочных действий пользователя с выдачей соответствующих сообщений.

Надежное функционирование разрабатываемого ПМК будет обеспечиваться при использовании современных ЭВМ, четком соблюдении рекомендаций. Запрещается удалять любые файлы проекта, доступ к ним должен быть ограничен.

Должно иметься резервное копирование таблиц для последующего их восстановления в случае необходимости.

Условия эксплуатации

Условия эксплуатации должны соответствовать санитарным и техническим нормам эксплуатации ЭВМ. Для работы с ПК допускаются работники, имеющие достаточный уровень знаний в предметной области. Для обслуживания данного программного комплекса нужен 1 человек.

Требования к составу и параметрам технических средств

Минимальные требования к программным и аппаратным средствам для нормального функционирования приложения:

Процессор: AMDилиIntel с частотой 1GHzи выше;

ОЗУ: 256 Mbи выше;

ОС: WindowsXP и выше;

Монитор: SVGA монитор;

Емкость ЖД: свободного места не менее 500 Mb;

Другие требования: сетевая карта, клавиатура, манипулятор мышь.

Требования к информационной и программной совместимости

Программная система функционирует в среде Windows XP и выше. Программный продукт создается с использованием инструментального средства разработки приложений C Sharp.

Требования к программной документации

Предварительный состав программной документации установлен в соответствии с ГОСТ 19.101-77. Ниже перечислен список программных документов и их содержание.

Текст программы – запись программы с необходимыми пояснениями и комментариями.

Описание программы – сведения о логической структуре и функционировании программы.

Программа и методика испытаний – требования, подлежащие проверке при испытании программы, также порядок и методы контроля.

Техническое задание – настоящий документ.

Пояснительная записка – схема алгоритма, общее описание алгоритма или функционирования программы, а также обоснование принятых технических и технико-экономических решений.

Эксплуатационные документы – описание применения, руководство пользователя.

Стадии и этапы разработки

Разработка ведется в несколько этапов, в соответствии с ГОСТ 19.101-77 и включает этапы, приведённые в таблице 1.5.

Таблица – Этапы разработки

Срок выполнения	
Техническое задание	Анализ и формализация требования к ПО, планирование работ
Эскизный проект	Предварительная разработка проекта ПО с использованием UML: диаграммы прецедентов использования, диаграммы классов и последовательности
Технический проект	Реализация рабочей версии ПО с основной функциональностью; тестирование
Рабочий проект	Корректировка и доработка программного обеспечения; разработка документации
Внедрение	Разработка мероприятий по внедрению и сопровождению ПО

Порядок контроля и приемки

ПМК автоматизации работы обработки и аппроксимации экспериментальных данных должен соответствовать требованиям заказчика и отвечать всем поставленным функциональным требованиям.

Контроль программного продукта осуществляется в следующем порядке.

–проверка функциональности разработанного ПО;

–проверка реакции программы на различные действия пользователя;

–проверка выходных данных;

–после выхода из программы операционная система должна продолжать работать корректно.

Принятие созданной системы заключается в тестировании его на рабочих местах после настройки программного продукта.

Практическое задание. Используя предметные области (см. список ниже), выполните следующие действия:

1. Определите, какие пункты будут необходимы в техническом задании на разработку вашего программного продукта.
2. Аргументированно объясните, почему возможен отказ от тех или иных пунктов ТЗ.
3. Выберите стандарты, которые необходимо будет учесть при разработке ПП.

Пользуясь приведёнными выше шаблонами, составьте техническое задание на разработку своего программного продукта.

Лабораторная работа №13. Анализ затрат на разработку программного продукта.

Теоретический материал.

1. Оценка затрат на разработку и внедрение программного обеспечения

1.1 Постановка проблемы

Подсчитывая затраты на разработку программного обеспечения, мы обычно рассуждаем следующим образом: «Когда мы разрабатывали нашу программу, нам пришлось купить лицензионное программное обеспечение, оплачивать аренду помещения, обновить наш компьютер, заплатить за свет и охрану помещения. Еще мы сами, наш директор и бухгалтер получали зарплату. Корме того, чтобы начать работу, мы были вынуждены взять кредит, рассчитываться за который нам придется в течение двух лет». Как оценить все затраты, необходимые для разработки и внедрения ПО? Можно сложить все перечисленные выплаты. Но параллельно с нашей программой фирма разрабатывала еще три. Да еще ремонтировала свой офис.

Для того чтобы разобраться во всем этом ворохе затрат начнем их классифицировать. Классификацию постараемся провести так, чтобы для каждого вида затрат имелись четкие методики их оценки, да и общие затраты вычислялись из частных по понятной методике.

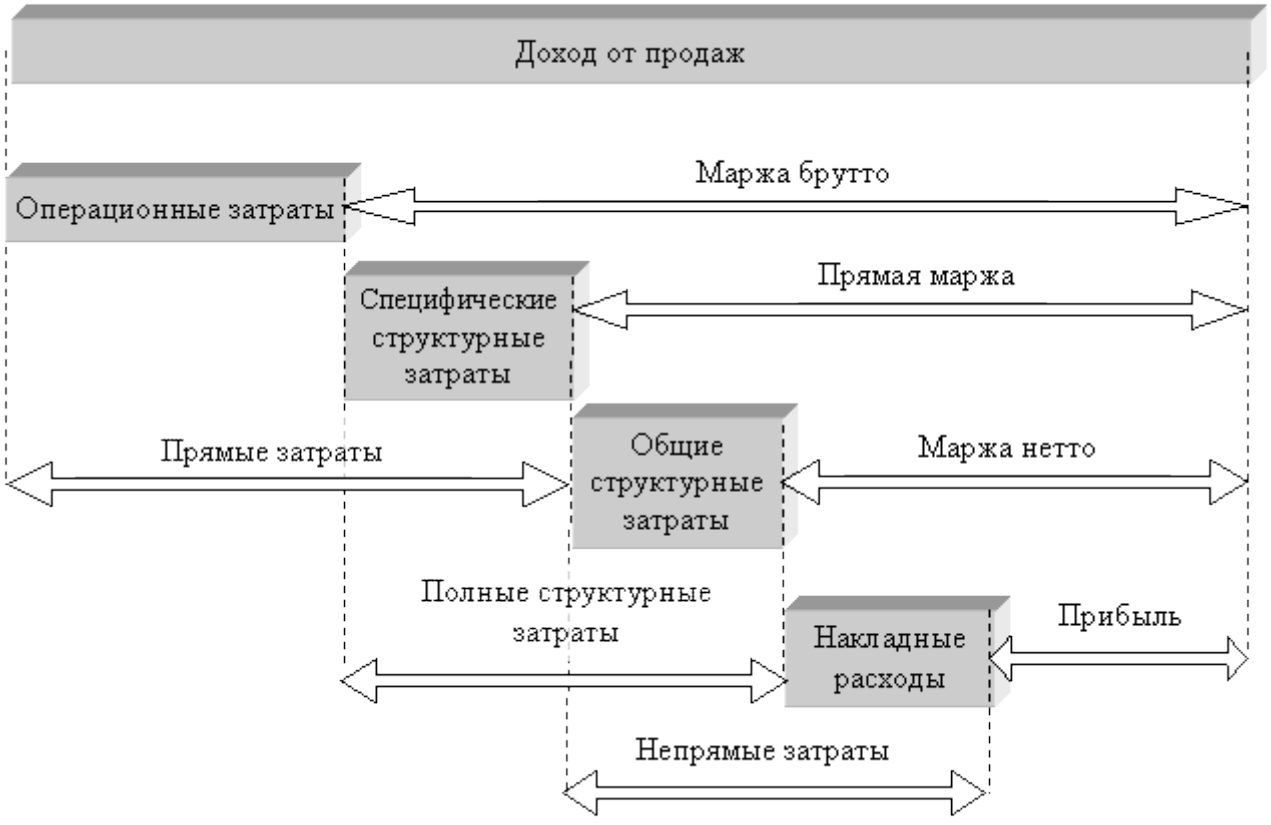
1.2 Классификация видов затрат. Маргинальный анализ

Количественная оценка величины затрат, как правило требует множества предположений. Каждое дополнительное предположение само нуждается в проверке и поэтому снижает достоверность результатов оценки. Поэтому, классифицируя затраты, сначала желательно выделить самые очевидные и легко рассчитываемые виды затрат. Отнимая эти затраты от ожидаемого дохода[1] мы на ранних этапах нашего исследования

можем оценить экономическую эффективность нашего проекта и отбросить неэффективные варианты его реализации. Разница дохода и затрат называется ***маржа,*** поэтому метод, основанный на анализе этой разницы называется «***Маргинальный анализ***». В настоящее время этот метод широко применяется для оценки эффективности проектов и направлений деятельности, а также в бизнес-планировании.

Изобразим весь доход, полученный от внедрения или продаж ПО, в виде длинного прямоугольника (рис. 1). На что тратится этот доход?

Прежде всего, выделим затраты зависящие от объема произведенной продукции. Назовем их ***операционными затратами***. Критерий выделения операционных затрат: они равны нулю, если данный продукт не производится вовсе. Пример. Затраты труда программистов зависят от объема и сложности разрабатываемой программы. И если мы разрабатываем две схожие программы, то труда на них потратим, скорее всего, в два раза больше, чем на одну программу. В таблице 1 приведены типичные примеры операционных затрат. Разница между доходом и операционными затратами называется грубой маржей или ***маржей брутто.*** Если маржа брутто слишком мала (или даже отрицательная), экономическая эффективность проекта явно не удовлетворительная.



Стоит рассмотреть альтернативные варианты проекта.

Рис. 1. Классификация затрат

Все другие затраты зависят от структуры предприятия, имеющегося у него оборудования, постоянного персонала, обязательств предприятия и т. д. Назовем их ***структурными затратами.***

Следующий по удобству выделения вид затрат – затраты, относящиеся к производству только данного продукта. Например, приобретение лицензионного ПО, используемого только нашей программой. Очевидно, что всю стоимость этого ПО нужно отнести на нашу программу. Назовем данный вид затрат ***специфическими структурными затратами.*** Особенностью двух первых видов затрат является то, что их напрямую можно отнести на производство данной продукции (проекта). Их сумма называется ***прямые затраты,*** а разность между доходом и прямыми затратами – ***прямой маржей.*** Прямая маржа – более точная оценка эффективности проекта.

Далеко не все затраты можно полностью отнести на один проект. Например, базовое программное обеспечение будет использоваться не только в нашем проекте, но и в других, поэтому его стоимость нельзя полностью отнести на один проект. Назовем такие затраты ***общими структурными затратами***. Для определения доли общих затрат, приходящихся на один проект, приходится делать предположения и договариваться о принципах распределения.

Наконец, само содержание фирмы требует затрат, не связанных напрямую с производством. К ним относятся затраты на содержание помещения, зарплату АУП и обслуживающего персонала, выплата налогов, штрафов и т. д. Такие затраты называются ***накладными расходами.*** Обычной практикой отнесения части накладных расходов на конкретный проект является определение процента. Бухгалтерия фирмы рассчитывает объем общефирменных расходов, делит его на общий объем производства и сообщает исполнителям, какую часть (процент) заработанных ими денег придется потратить на общефирменные нужды.

Остаток от вычитания из дохода всех видов затрат – это ***прибыль***. Наличие прибыли свидетельствует о рентабельности (экономической состоятельности) проекта.

1.3 Методики расчета различных видов затрат

Приведенная выше классификация затрат позволяет применить относительно простые методики для их оценки. Рассмотрим особенности расчетов каждого вида затрат. Для наглядности последовательность расчета проиллюстрирована примерами расчетов затрат на дипломный проект. При этом следует помнить, что дипломный проект – учебная работа. Выполняя его, студент бесплатно пользуется вычислительными средствами института, его библиотекой и т. д. При этом он сам не получает заработную плату. Но если мы не учтем перечисленные виды затрат оценка эффективности проекта будет искаженной. Более реалистичной будет оценка в предположении, что проект выполняется на предприятии, которое оплачивает все виды услуг. При этом считается, что оплата труда студента как у начинающего программиста, т. е. равна средней оплате выпускника института.

1.3.1 Операционные затраты

Это наиболее простой для расчета вид затрат. В большинстве практически важных случаев затраты вычисляются как произведение стоимости единицы ресурса на количество потребленного ресурса. В таблице 1 сведены основные виды операционных затрат, встречающихся при разработке программного обеспечения.

Таблица 1

Виды операционных затрат	**Методика расчета**
Зарплата[2] при повременной оплате труда	Затраты = (фактическое время) *(повременная ставка)*(1 + % отчислений)
Материалы	Затраты = (количество использованных материалов)*(стоимость единицы)
Услуги	Затраты = (объем услуг) *(стоимость единицы услуги)

Пример расчета операционных затрат

Разработка дипломного проекта длится 4 месяца. В ней участвуют студент–дипломник и его руководитель. Для выполнения дипломной работы потребовалось:

использовать Internet в течение 4 месяцев с общим трафиком 150 Mbt; приобрести:

3 CD- болванки для создания инсталляционных дисков; 3,5’ дискет для работы; пачку бумаги для оформления дипломных проектов; других канцелярских принадлежностей на сумму 150 руб.

Запишем исходные данные и результаты в таблицу 2

Таблица 2

Операционные затраты

Вид затрат	**Ед. измерения**	**Стоимость ед., руб.**	**Затраты**	**Стоимость руб.**
Труд студента[3]	чел. мес.	2730	4	10920
Труд руководителя[4]	чел. мес.	4095	1,5	6142,5
Абонентская плата за Internet	мес.	200	4	800
Трафик Internet	Mbt	7,44	150	1116
Болванки CD	шт	12	3	36

3,5’ дискеты	шт	10	10	100
Бумага для принтера	пачка	200	1	200
Канцелярские принадлежности	-	150	1	150
Итого	1946 4,5			

Общие операционные затраты составили (округленно) 19,5 тыс. руб.

1.3.2 Специфические структурные затраты

Затраты на оборудование

Как правило, оборудование, приобретенное для выполнения проекта, будет использоваться и после его завершения. Поэтому полностью относить его стоимость на один проект нельзя. Для определения доли стоимости оборудования, отнесенной к конкретному проекту, используется метод ***начисления амортизации***.

Определяется ресурс оборудования (например, в годах работы). Общая стоимость оборудования делится на ресурс. Результат – стоимость единицы ресурса. Определяется объем ресурса оборудования, необходимый для реализации проекта. Доля стоимости ресурса, отнесенная на проект равна стоимости единицы ресурса, умноженной на необходимый объем.

Средства вычислительной техники устаревают достаточно быстро. Основная причина этого – технический прогресс. Новые поколения техники настолько превосходят предыдущие, что использовать технику 2-3 летней давности уже не имеет никакого смысла. Поэтому для вычислительной техники назначаются сокращенные сроки амортизации (2-3 года.)

Пример расчета затрат на оборудование**.**

При разработке проекта используются: два компьютера Pentium 4, принтер HP LJ 1200 и сканер Epson Perfection 1260 PHOTO. Продолжительность проекта – 6 месяцев. Компьютеры и принтер используются непрерывно, а сканер только во второй половине проекта. Сведем все данные в таблицу 3 и проведем необходимые расчеты.

Таблица 3

Затраты на оборудование

Вид оборудования	**Стоимость, руб.**	**Период амортизации, лет**	**Стоимость ед. ресурса, руб.**	**Требуемый объем, лет**	**Доля, отнесенная на проект, руб.**

Компьютер Pentium 4	17850	3	5950	1	5950
Принтер HP LJ 1200	19500	2	9750	0,5	4875
Сканер Epson Perfection 1260 PHOTO	4397	3	1466	0,25	366
Итого	11191				

Стоимость оборудования, отнесенная на проект равна (округленно) 11200 руб.

1.4 Приведение затрат к одному времени

От начала разработки проекта до его внедрения проходит определенное время. Если даже проект рентабельный, деньги, затраченные на его реализацию, будут возвращаться только после внедрения проекта. Таким образом, деньги, вложенные в проект как бы «омертвляются». Они не могут быть использованы ни на что другое до тех пор, пока проект не будет внедрен. А жизнь идет своим чередом: происходит инфляция, подворачиваются другие заманчивые проекты, от которых приходится отказаться, так как нет денег. Таким образом, рубль, вложенный в начале проекта и рубль, полученный после его реализации, оказываются не эквивалентными. Если проект достаточно продолжительный, не эквивалентными будут рубли, вложенные на разных этапах реализации проекта. Как учесть эту неэквивалентность? Обычно для этих целей используется методика приведения к единовременным затратам.

Для сопоставления разновременных затрат рассмотрим два различных варианта их использования:

1) Мы вкладываем деньги **C0** руб. в проект и готовы ждать до его внедрения **t** лет.

2) Вместо вложения в проект, мы положим деньги в банк на те же **t** лет под **P** процентов годовых.

Положив деньги в банк, по истечении года, мы получим:

C1= C0*(1+ P) руб.

После двух лет, учитывая сложные проценты, получаем:

C2= C1*(1+ P) = C0*(1+ P) *(1+ P) руб.

Через **t** лет наш вклад составит:

Ct= C0*(1+ P)t

Таким образом, во втором варианте, рубль, который мы имеем сейчас будет эквивалентен **(1+ P)t** рублям. Можно предположить, что и в первом варианте через **t** лет рубль подешевеет в **(1+ P)t** раз.

В качестве **Р** в своих расчетах мы использовали банковский процент, то есть долю, на которую банк обещает вкладчику увеличить сумму его вклада через год. Назначая банковский процент, банки ориентируются на состояние экономики страны, перспективы ее развития, инфляцию и т. д. Поэтому величина банковского процента достаточно хорошо отражает мнения специалистов и может быть использована для надежного предсказания будущего состояния экономики.

Сильно ли влияет приведение к единовременным затратам на оценку эффективности проекта? Рассмотрим пример. Для реализации проекта необходимо приобрести установку стоимостью 100 тыс. руб. Когда это сделать? Проект длится полгода и его внедрение займет еще полгода. Текущий банковский процент равен 20% годовых. Установку можно купить в начале проекта, но использовать ее можно только тогда, когда для нее написано программное обеспечение, т. е. через полгода. Сопоставим затраты по каждому варианту, приведенные к одному моменту времени (полгода после начала проекта):

1 вариант: С=100*(1+0,20)0,5 = 109,54 тыс. руб.

2 вариант: С=100,00 тыс. руб.

Разница 9,54 тыс. руб.

1.5 Затраты на нематериальные активы

Разработка программного обеспечения связана с использованием лицензионных операционных систем, программных сред и утилит, графических изображений и фонтов. Все они являются объектами интеллектуальной собственности и их нелицензионное («пиратское») использование незаконно. Приобретение прав на использование таких объектов не сопровождается передачей покупателю каких либо материальных объектов. Поэтому, в бухгалтерском учете права на использование объектов интеллектуальной собственности регистрируются как «Нематериальные активы» [3].

Как учесть затраты на объекты интеллектуальной собственности?

Если они входят в специфические структурные затраты (т. е. не используются ни где, кроме нашего проекта), вся их стоимость относится на стоимость проекта.

Если они используются несколькими проектами, или планируется их использование после окончания проекта, применяется метод ***начисления амортизации.*** Права на использование объектов интеллектуальной собственности могут быть бессрочными или ограниченными во времени [4]. Период амортизации, естественно не может превышать срок прав (использование объекта по истечении срока незаконно). Однако период амортизации может быть меньше этого срока (например, в связи с моральным устареванием используемого объекта интеллектуальной собственности).

Пример.

При разработке проекта используются следующие виды лицензионного обеспечения (Таблица 4)

Таблица 4

Затраты на лицензии

Объекты интеллектуальной собственности	**Стоимость лицензии, руб.**	**Период амортизации, лет**	**Стоимость ед. ресурса**	**Требуемый объем**	**Доля, отнесенная на проект**
Операционная система Windows XP	11100	3	3700	1	3700
Программная среда Builder C++	31800	2	15900	0,5	7950
Пакет Microsoft Office	6400	3	2100	0,25	525
Пакет Fine Reader	4800	3	1600	0,25	400
ИТОГО	**12575**				

1.6 Общефирменные затраты и накладные расходы

Если прямые затраты разработчик в большинстве случаев может рассчитать самостоятельно, то для оценки непрямых затрат ему придется обратиться за помощью в бухгалтерию фирмы. Для упрощения такой оценки, обычно поступают следующим образом. Подсчитав общий объем затрат на содержание фирмы и сопоставив их с ожидаемым доходом, бухгалтерия определяет долю (процент) непрямых затрат в стоимости работ, выполняемых фирмой. Заключая договора, разработчики увеличивают стоимость работ на указанный процент.

Например. Рассчитав затраты на: содержание и ремонт помещения, оплату труда администрации и вспомогательному персоналу, выплату процента по кредитам, уплату налогов (за исключением уже учтенного нами налога с заработной платы), бухгалтерия установила, что непрямые расходы фирмы должны составлять 40% от ожидаемого дохода.

Соберем все расчеты по нашему примеру в таблицу 5.

Таблица 5

Смета затрат

Вид затрат	**Объем, руб.**

Оплата труда	12 500
Налог на фонд З. П.	4 563
Услуги Internet	385
Приобретение материалов	486
Амортизация основных производственных фондов	11 191
Амортизация объектов интеллектуальной собственности	12 575
Всего прямых затрат	41 700
Общефирменные затраты и накладные расходы	16 680
ИТОГО	58 380

Расчеты затрат достаточно просты, но громоздки. Основное условие их проведение четкость предположений, на которых базируются исходные данные и выбор методов, а также аккуратность расчетов. Для подобных расчетов удобно применять электронные таблицы.

Практическое задание. Используя предметные области (см. список ниже), выполните следующие действия:

1. Определите, какие затраты будут очевидно необходимы в вашем случае при разработке программного продукта, и составьте их полный перечень.
2. Аргументированно объясните, почему возможен отказ от тех или иных затрат.
3. Рассчитайте операционные затраты, пользуясь вышеприведёнными методиками.

Рассчитайте структурные затраты и накладные расходы, пользуясь вышеприведёнными методиками.

Лабораторная работа №14. Составление эскизного проекта.

Теоретический материал.

Эскизный проект

Эскизным проектом называют пакет конструкторской документации, создаваемый на стадии разработки автоматизированной системы. Цель создания этих документов – установить принципиальные, конструктивные решения, представить их для ознакомления с принципами работы и устройством разрабатываемой системы. Также этим проектом может рассматриваться несколько вариантов устройства АСУ.

Эскизный проект на автоматизированную систему разрабатывают перед техническим проектом или вместе с ним. Эта документация может и не оформляться в случае, если ею не может быть предоставлено никаких новых данных – ее необходимость устанавливается тех заданием.

Разработка эскизного проекта

Эскизный проект программного продукта или автоматизированной системы разрабатывается в соответствии со следующими этапами:

- **Декомпозиция АС на комплексы и проработка архитектурных решений**
На этом этапе выбирается уровень автоматизации. Анализируются аналогичные системы. Выбирается схема взаимодействия оператора с системой, оцениваются алгоритмы и процессы обработки данных. По итогам этого этапа разработки эскизного проекта получают более точные требования к АС, а также функциональную внешнюю спецификацию ее комплексов

- **Разработка операционной среды или требований к ней**
На этом этапе составляются требования к конфигурации оборудования и операционной среды, выбирается операционная система

- **Разработка методов анализа качества системы**, с соответствующими метриками показателей качественной оценки, необходимыми для проведения испытаний

- **Планирование перспектив создания АСУ**

На данном этапе выбираются основные концепции технологий разработки, уточняются требования к программным средствам, создается база данных по автоматизации и контролю работ, а также проходит оформление пояснительной записки.

Перечень документов эскизного проекта

Эскизный проект на автоматизированную систему, согласно ГОСТ 34.201-89 состоит из такой документации как:

Ведомость ЭП

Пояснительная записка

Структурная схема комплекса техсредств

Функциональная схема структуры автоматизированной системы управления

Перечень заданий, составленных для разработки специальных дополнительных техсредств

Схема автоматизации

Технические задания по разработке дополнительных тех. средств.

Эскизный проект может не содержать некоторых документов, приведенных в данном списке, а также может быть дополнен иной документацией. Полный список документов выбирается в зависимости от специфики конкретной системы, требований заказчика и регламентируется техническим заданием.

Ниже приведён пример эскизного проекта.

Пример эскизного проекта

УТВЕРЖДАЮ

Руководитель (заказчика ИС)

Личная подпись Расшифровка подписи __

Печать

Дата « » 2020 г.

УТВЕРЖДАЮ

Руководитель (разработчика ИС)

Личная подпись Расшифровка подписи

Печать

Дата « » 2020 г.

Эскизный проект на создание информационной системы

Система Управления Базой Данных (наименование вида И С)

АВТОСТОЯНКА «Название»

(наименование объекта информатизации) СУБД «Автостоянка»

(сокращенное наименование И С)

На N листах

Действует с « » 2020 г.

Содержание

Решения по структуре системы

Решения по режимам функционирования работы системы

Решения по численности, квалификации и функциям персонала АС.

Состав функций комплексов задач, реализуемых системой

Решения по составу программных средств, языкам деятельности, алгоритмам процедур и операций и методам их реализации

Источники разработки

Ведомость эскизного проекта

На предыдущих стадиях разработки СУБД «Автостоянка» были составлены и утверждены следующие документы:

• Техническое задание на создание информационной системы СУБД «Автостоянка», разработанное на основании ГОСТ 34.602—89 на написание ТЗ на автоматизированные системы управления от г.2020

Пояснительная записка к эскизному проекту

Общие положения

Данный документ является эскизным проектом на создание Системы Управления Базой Данных для Автостоянки «Название» (СУБД «Автостоянка»)

Перечень организаций, участвующих в разработке системы, сроки и стадии разработки, а также ее цели и назначение указаны в техническом задании на создание информационной

системы.

Основные технические решения

Решения по структуре системы

СУБД «Автостоянка» будет представлять собой персональную систему управления локальной базой данных, работающей на одном компьютере. Система будет управлять реляционной базой данных, представляющей собой набор связанных между собой таблиц в формате MS SQL Server, доступ к которым осуществляется с помощью ключей или индексов. Сведения в одной таблице могут отражать сведения из другой, и при изменении сведений в первой таблице эти изменения немедленно отображаются во второй. Таким образом будет достигнута непротиворечивость данных.

Общая структура базы данных:

• Информация о пользователе автостоянки:

— ФИО.

— Адрес проживания.

— Марка автомобиля.

— Дата и время въезда.

— Период пользования.

• Информация об оплате:

— Платежные реквизиты:

— История выплат

— Информация о наличии задолженностях

— Информации о наличии скидок

Автоматизированная система должна выполнять следующие функции:

• сделать запись о владельце автомобиля;

• редактировать информацию о владельце автомобиля;

• удалить информацию о владельце автомобиля;

• внести информацию об оплате;

• редактировать информацию об оплате;

• удалить информацию об оплате;

Решения по составу программных средств, языкам деятельности, алгоритмам процедур и операций и методам их реализации

Для реализации АС будет использоваться среда программирования MS Visual Studio 2019 и язык программирования C# 5.0.

Источники разработки

Данный документ разрабатывался на основании ГОСТ 34.698—90 на написание ТЗ на автоматизированные системы управления от г. 2020

СОСТАВИЛИ

Должность исполнителя

Фамилия, имя, отчество

Подпись

Дата « » 2020 г.

Практическое задание. Используя предметные области (см. список ниже), выполните следующие действия:

1. Составьте схему эскизного проекта.
2. Определите, какие пункты могут вам потребоваться, а какие – нет.

Составьте эскизный проект, пользуясь приведённым в теоретической части шаблоном.

Лабораторная работа №15. Исследование методик подведения промежуточных итогов на разных этапах разработки ПО.

Теоретический материал.

Спринт — это короткий временной интервал, в течение которого scrum-команда выполняет заданный объем работы. Спринты лежат в основе методологий scrum и agile, и правильный выбор спринтов поможет вашей agile-команде выпускать более качественное программное обеспечение без лишней головной боли.

«При использовании scrum продукт разрабатывается в ходе нескольких итераций с фиксированной продолжительностью, которые называются спринтами и разбивают большие сложные проекты на небольшие задачи», — говорит Меган Кук, менеджер группы товаров для Jira Software в Atlassian.

«Спринты повышают управляемость проектов, позволяют командам выпускать высококачественные продукты быстрее и чаще, а также обеспечивают большую гибкость для адаптации к изменениям».

Многие ассоциируют scrum-спринты с agile-разработкой программного обеспечения настолько часто, что scrum и agile принимают за синонимы. Однако это не так. Agile — это набор принципов, а scrum — методика для активного решения задач.

Многочисленные сходства между глобальными задачами agile и процессами scrum вполне справедливо приводят к тому, что эти два понятия ассоциируются друг с другом. Благодаря спринтам команды могут следовать agile-принципу «частой поставки рабочего программного обеспечения», а также реализовать agile-задачу «реагирования на изменения в соответствии с планом». Установки scrum — прозрачность, проверка и адаптация — дополняют agile-методику и играют главную роль в концепции спринтов.

Руководство по scrum закладывает прочную теоретическую основу для данного обсуждения спринтов. Наша цель — внести немного красок в эту тему, открыв лучшие практики от людей, которые занимаются этой работой каждый день.

Как планировать и выполнять спринты в scrum

Люди, разработавшие scrum, действительно все предусмотрели. Чтобы запланировать предстоящий спринт, нужно провести собрание по планированию спринта. Планирование спринта — это мероприятие, на котором команда сообща отвечает на два основных вопроса: какую работу можно выполнить в этом спринте и как эта работа будет выполняться?

Выбором подходящих рабочих задач для спринта занимаются совместно владелец продукта, scrum-мастер и команда разработчиков. Владелец продукта определяет цель, которая должна быть достигнута в этом спринте, и задачи из бэклога продукта, при выполнении которых будет достигнута эта цель.

Затем команда создает план, согласно которому будут выполняться задачи бэклога, чтобы к окончанию спринта вся работа была завершена. Выбранные рабочие задачи и план по их выполнению называется бэклогом спринта. К концу совещания по планированию спринта команда готова приступить к работе. Для этого необходимо просто выбирать задачи из бэклога спринта и менять их статус с «В работе» на «Готово» по мере завершения работы.

В течение спринта команда собирается на ежедневные scrum-совещания, или стендапы, чтобы обсудить ход выполнения работы. Цель такого совещания заключается в выявлении блокеров и проблем, которые могут повлиять на достижение цели спринта.

По окончании спринта команда показывает выполненную работу на обзоре итогов спринта. Здесь можно продемонстрировать итоги работы заинтересованным сторонам и другим участникам команды до того, как они попадут в рабочую среду.

Завершите цикл спринтов на моем любимом собрании — ретроспективе спринта. Здесь команда может определить области, требующие улучшения в следующем спринте. С этими сведениями можно начинать следующий цикл спринта.

Что стоит и не стоит делать

Даже если основы уже известны, большинство команд спотыкается в начале работы со спринтами. Меган Кук завершает эту дискуссию списком действий, которые стоит и не стоит делать при использовании спринтов, которые она сформулировала за годы своей работы.

Что стоит делать.

- Убедитесь, что команда понимает цель спринта и способ измерения успеха. Это ключ к тому, чтобы все участники настроились и двигались к общему месту назначения.
- Убедитесь, что у вас есть четкий и понятный бэклог с приоритетами и зависимостями. Плохо управляемый бэклог может превратиться в большую проблему и сорвать процесс работы.
- Убедитесь, что вы хорошо представляете скорость работы команды и учитываете такие события, как отпуска и общие собрания.
- Используйте собрание по планированию спринта, чтобы расширить описание работы, которую необходимо выполнить, дополнительными подробностями. Поощряйте участников команды за наброски общего плана для всех историй, ошибок и задач, которые входят в спринт.
- Пропускайте работу там, где вы не сможете получить связанные результаты, например работу другой команды, дизайн и юридическое подтверждение.
- Наконец, после принятия решения или составления плана убедитесь, что есть человек, который фиксирует эту информацию в инструменте управления проектами или инструменте для совместной работы, например в заявках Jira. В таком случае позднее каждый сможет легко увидеть и решение, и обоснование.

И если уж вы работаете над тем, чтобы стать сильным специалистом по scrum, выполняя рекомендации, ознакомьтесь также с действиями, которые выполнять не следует.

Что не стоит делать.

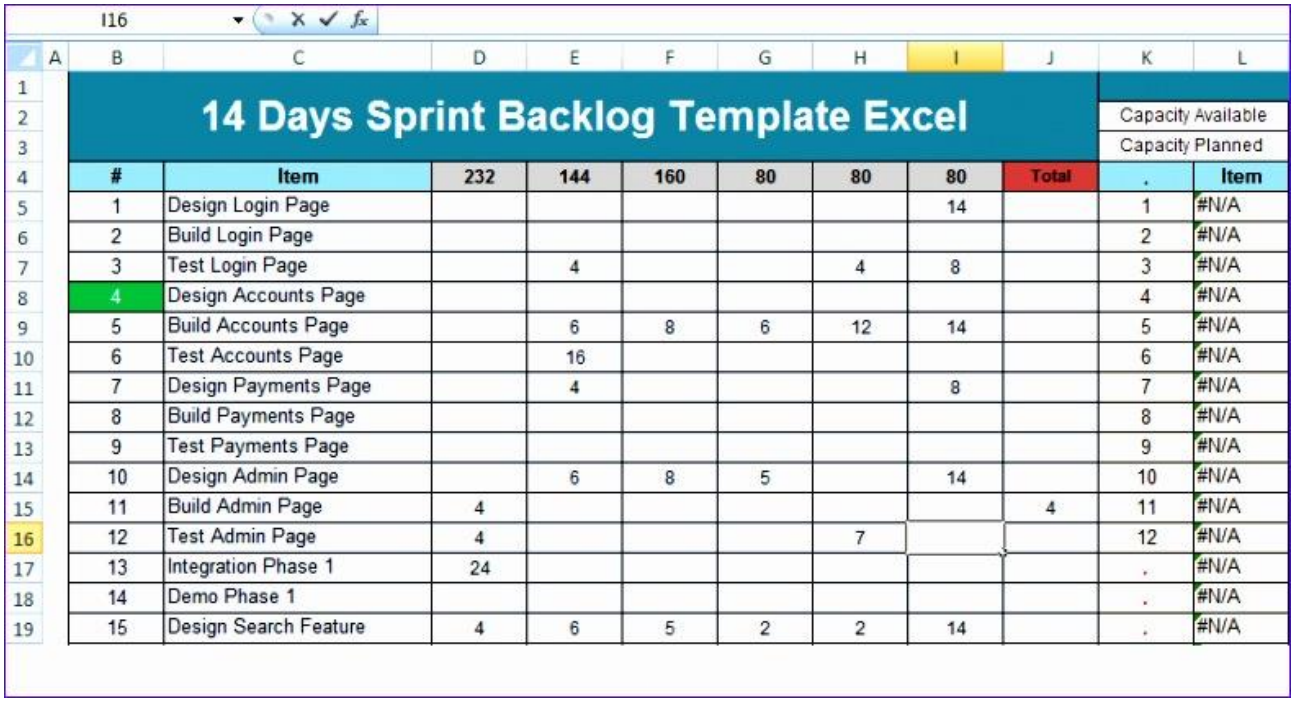
- Не берите слишком много историй, не переоценивайте скорость работы и не выполняйте задачи, которые не могут быть выполнены в этом спринте. Вы же не хотите, чтобы вас или вашу команду постигла неудача?
- Не забывайте о качестве или техническом долге. Удостоверьтесь, что у вас есть время для проведения контроля качества и работы, не связанной с функционалом, например исправления ошибок и технологического контроля.
- Не допускайте, чтобы команда имела смутное представление о содержимом спринта. Определите объем работ и не зацикливайтесь слишком сильно на *скорости* продвижения; убедитесь, что все участники команды работают в *одном направлении*.
- Кроме того, не берите слишком большое количество неопределенной или высокорисковой работы. Делите крупные истории или истории с высокой степенью неопределенности и не бойтесь оставлять часть этой работы для следующего спринта.

- Если команда выражает обеспокоенность по поводу скорости, уровня неопределенности в работе или слишком большого объема работы, не игнорируйте их мнение. Рассмотрите эту задачу и внесите коррективы при необходимости.

Подробнее о спринтах

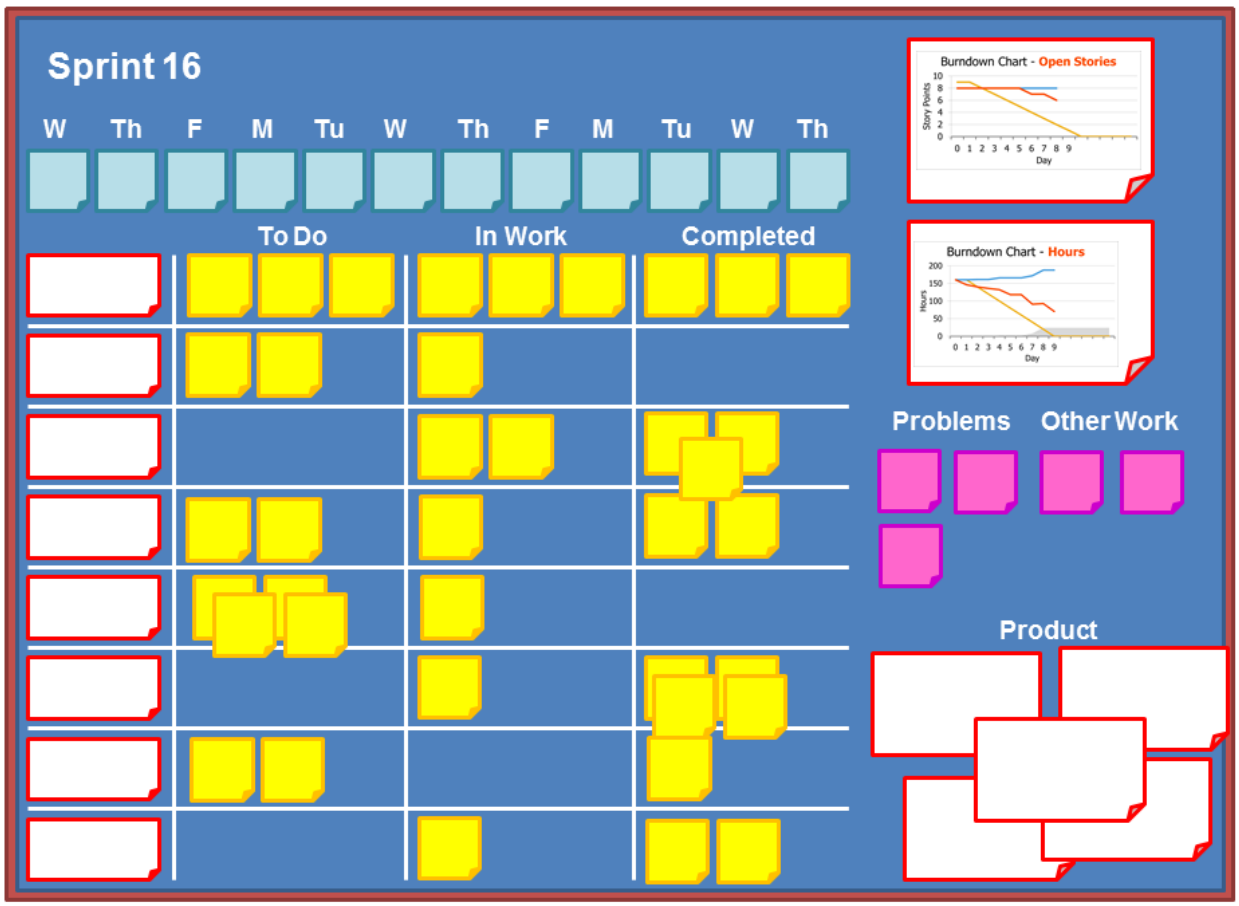
Спринты настолько известны (и настолько эффективны), что их часто считают первым шагом на пути к повышению гибкости. Но мы выяснили, для освоения спринтов необходимо овладеть некоторыми scrum- и agile-понятиями, которые строятся друг на друге.

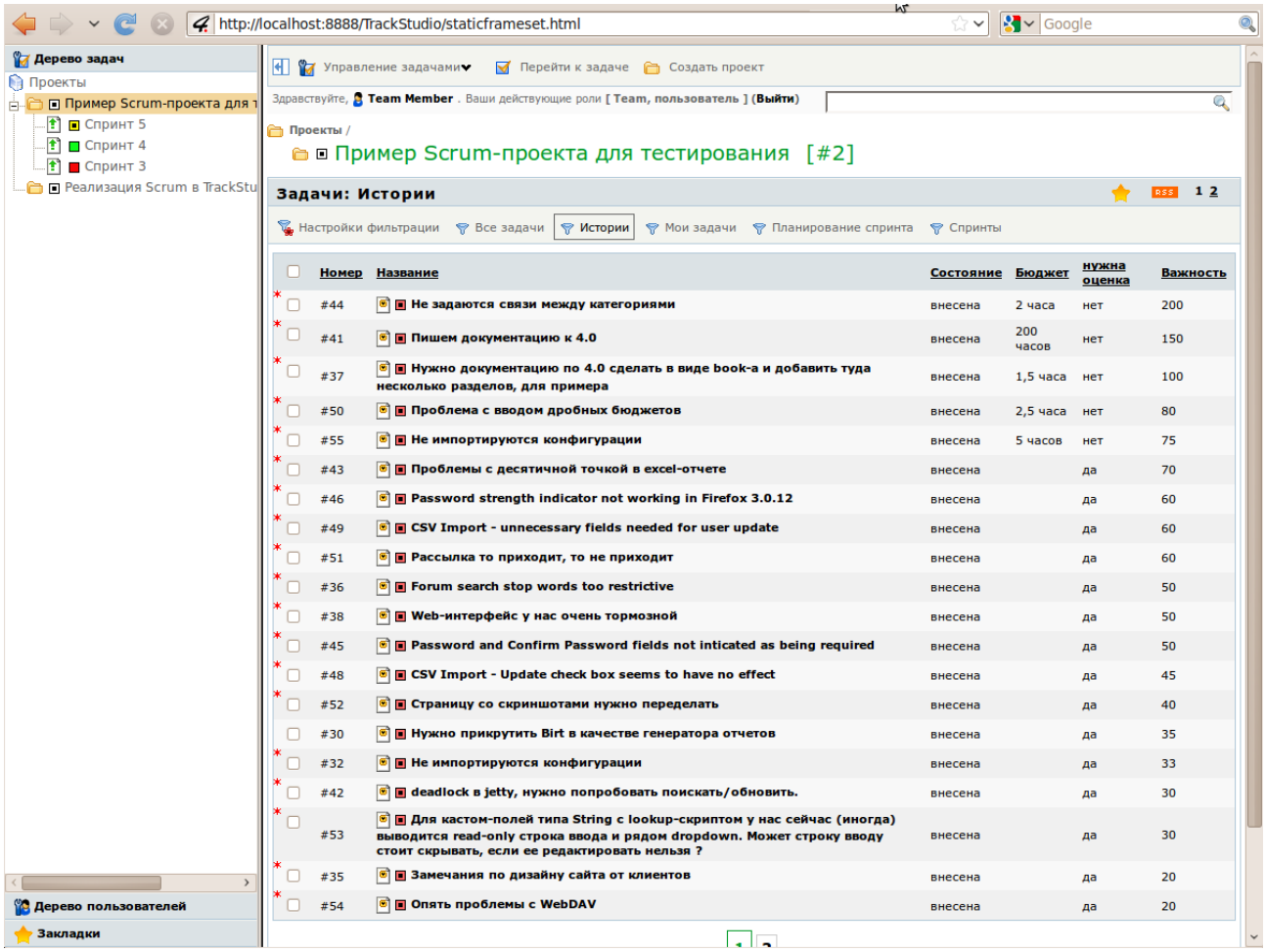
Пример спринта:



I16

	A	B	C	D	E	F	G	H	I	J	K	L
1		14 Days Sprint Backlog Template Excel										
2											Capacity Available	
3											Capacity Planned	
4		#	Item	232	144	160	80	80	80	Total	.	Item
5		1	Design Login Page						14		1	#N/A
6		2	Build Login Page								2	#N/A
7		3	Test Login Page		4			4	8		3	#N/A
8		4	Design Accounts Page								4	#N/A
9		5	Build Accounts Page		6	8	6	12	14		5	#N/A
10		6	Test Accounts Page		16						6	#N/A
11		7	Design Payments Page		4				8		7	#N/A
12		8	Build Payments Page								8	#N/A
13		9	Test Payments Page								9	#N/A
14		10	Design Admin Page		6	8	5		14		10	#N/A
15		11	Build Admin Page	4						4	11	#N/A
16		12	Test Admin Page	4				7			12	#N/A
17		13	Integration Phase 1	24							.	#N/A
18		14	Demo Phase 1								.	#N/A
19		15	Design Search Feature	4	6	5	2	2	14		.	#N/A





Номер	Название	Состояние	Бюджет	нужна оценка	Важность
#44	Не задаются связи между категориями	внесена	2 часа	нет	200
#41	Пишем документацию к 4.0	внесена	200 часов	нет	150
#37	Нужно документацию по 4.0 сделать в виде book-а и добавить туда несколько разделов, для примера	внесена	1,5 часа	нет	100
#50	Проблема с вводом дробных бюджетов	внесена	2,5 часа	нет	80
#55	Не импортируются конфигурации	внесена	5 часов	нет	75
#43	Проблемы с десятичной точкой в excel-отчете	внесена		да	70
#46	Password strength indicator not working in Firefox 3.0.12	внесена		да	60
#49	CSV Import - unnecessary fields needed for user update	внесена		да	60
#51	Рассылка то приходит, то не приходит	внесена		да	60
#36	Forum search stop words too restrictive	внесена		да	50
#38	Web-интерфейс у нас очень тормозной	внесена		да	50
#45	Password and Confirm Password fields not inticated as being required	внесена		да	50
#48	CSV Import - Update check box seems to have no effect	внесена		да	45
#52	Страницу со скриншотами нужно переделать	внесена		да	40
#30	Нужно прикрутить Birt в качестве генератора отчетов	внесена		да	35
#32	Не импортируются конфигурации	внесена		да	33
#42	deadlock в jetty, нужно попробовать поискать/обновить.	внесена		да	30
#53	Для кастом-полей типа String с lookup-скриптом у нас сейчас (иногда) выводится read-only строка ввода и рядом dropdown. Может строку вводу стоит скрывать, если ее редактировать нельзя ?	внесена		да	30
#35	Замечания по дизайну сайта от клиентов	внесена		да	20
#54	Опять проблемы с WebDAV	внесена		да	20

Пример выполнен в Track Studio.

Практическое задание. Используя предметные области (см. список ниже) и ваши предыдущие наработки, выполните следующие действия:

1. Разбейте весь процесс разработки ПО на отдельные этапы.
2. Определите продолжительность этих этапов.

Составьте спринты на каждый из этих этапов.

Лабораторная работа №16. Распределение ролей при разработке ПП.

Теоретический материал.

Владелец продукта (Product Owner)

Цель:

Позиционирование и продвижение продукта на рынке, достижение бизнес целей.

Задачи:

1. Определение концепции продукта;
2. Создание Go To Market стратегии;
3. Сегментация и анализ рынка, определение ценности;
4. Анализ конкурентов;
5. Управление списком задач (бэклогом) и пиритизация требований;
6. Контроль статуса разработки;
7. Выбор продуктовой стратегии и методов монетизации;
8. Генерация гипотез по улучшению бизнес показателей;
9. Оценка достижения бизнес показателей;
10. Построение процесса / цикла обратной связи от пользователей и оценки качества продукта

Управляющий проектом (Project Manager)

Цель:

Разработка продукта в срок, не превышая выбранный бюджет и с надлежащим качеством.

Задачи:

1. Управление командой (формирование, мотивация, контроль);

2. Создание RoadMap (плана разработки);
3. Оценка стоимости разработки;
4. Создание и распределение задач, контроль выполнения;
5. Организация командных активностей;
6. Проведение интервью и встреч с заказчиком;
7. Решение организационных вопросов;
8. Участие в приемке продукта;
9. Прием решений по сложным вопросам (всем);
10. Прием решений о публикации новой версии системы (совместно с Техническим лидером);

Бизнес-аналитик (Business Analyst)

Цель:

Создание и оптимизация бизнес процессов для достижения целей бизнеса

Задачи:

1. Разработка концепции программного продукта;
2. Определение ролей пользователей и их потребностей;
3. Описание предметной области (в т.ч. основных объектов и связей между ними), создание и оптимизация бизнес процессов
4. Управление требованиями к ПО;
5. Проведение интервью с заказчиками и конечными пользователями;
6. Анализ конкурентов;
7. Консультация команды разработки;
8. Участие в приемке продукта и анализ поведения пользователей;
9. Оценка стоимости разработки (совместно с PM);

Системный аналитик (System Analyst)

Цель:

Обеспечение эффективной работы системы для успешного выполнения целей пользователей.

Задачи:

1. Определение ролей пользователей и их потребностей (если этого не делает BA);

2. Описание предметной области (в т.ч. основных объектов, их атрибутов, связей между ни ми), бизнес процессов, потоков данных;
3. Управление требованиями к ПО;
4. Разработка прототипов и UX (совместно с дизайнером);
5. Проведение интервью с заказчиками и конечными пользователями;
6. Формирование стека задач (бэклога) (если этого не делает PM);
7. Консультация команды разработки;

Аналитик данных (Data Scientist)

Цель:

Выявление скрытых закономерностей в данных для оптимизации бизнес процессов.

Задачи:

1. Фиксация бизнес показателей;
2. Организация сбора данных и мониторинга показателей;
3. Построение моделей;
4. Проверка гипотез по улучшению бизнес показателей;
5. Составление отчетов.

Системный архитектор (System Architect)

Цель:

Проектирование архитектуры системы, удовлетворяющей требованиям (как к функциям системы, так и нагрузкам на систему)

Задачи:

1. Разработка архитектуры системы и выбор стека технологий;
2. Контроль за соблюдением рекомендаций по архитектуре;
3. Прием сложных технических решений;
4. Консультация команды разработки;

Технический лидер (TechLead)

Цель:

Координация технической команды.

Задачи:

1. Создание и распределение технических задач, контроль выполнения;
2. Консультация программистов по узкотехническим вопросам;
3. Ревью кода;
4. Прием решений о публикации новой версии системы (совместно с ПМ);
5. Публикация системы в сторах;
6. Оценка стоимости разработки (совместно с РМ);

Программист (Programmer)

Цель:

Разработка программной системы в соответствии с поставленными требованиями.

Задачи:

1. Разработка программной системы (написание кода, разработка структуры базы данных и т.д.);
2. Принятие решений о способе разработки;
3. Контроль качества разработки и проведение ревью кода;
4. Тестирование кода;
5. Написание технической документации;
6. Выпуск новой версии продукта.

Специалист по качеству (тестировщик) (QA)

Цель:

Минимизация ошибок в работе системы.

Задачи:

1. Тестирование требований;
2. Написание тест кейсов и тест планов;
3. Тестирование системы (регрессионное, нагрузочное, функциональное и т.д.);
4. Разработка авто-тестов;
5. Поиск багов на основе отзывов от пользователей.

Дизайнер интерфейсов (UI/UX Designer)

Цель:

Разработка удобного и привлекательного интерфейса пользователя программной системы.

Задачи:

1. Разработка дизайнов экранов;
2. Разработка ScreenFlow;
3. Разработка дизайн-концепта и гайдлайнов;
4. Разработка прототипов экранов;
5. Оптимизация пользовательского взаимодействия (создание рекомендаций);
6. Консультация команды разработки.

Технический писатель (TechWriter)

Цель:

Разработка пользовательской и технической документации.

Задачи:

1. Разработка пользовательской документации;
2. Разработка FAQ;
3. Разработка описания API.

Специалист тех. поддержки (TechSupport)

Цель:

Минимизация недовольства пользователей ПО за счет помощи и ответов на вопросы.

Задачи:

1. Ответы на вопросы пользователей;
2. Решение проблем пользователей;
3. Сбор обратной связи от пользователей;
4. Фиксация багов, найденных пользователями.

Системный администратор/DevOps (System Administrator)

Цель:

Минимизация технических ошибок при эксплуатации системы, аппаратно-программная поддержка команды разработки

Задачи:

1. Обслуживание и анализ загрузки серверов;
2. Создания необходимых условий в инфраструктуре для нормального функционирования ПО;
3. Помощь в развертывании системы и настройка среды для быстрой публикации новых версий;
4. Контроль логов;
5. Настройка инструментов для автоматизации процесса разработки и тестирования.

Маркетолог (Marketing Specialist)

Цель:

Продвижение продукта (программной системы) на рынке.

Задачи:

1. Анализ рынка;
2. Продвижение сайта продукта, SEO;
3. Помощь в публикации программной системы в сторах
4. Подготовка PR материалов;
5. Продвижение в соц. сетях;
6. Организация и проведение рекламных компаний.

Практическое задание. Используя предметные области (см. список ниже), выполните следующие действия:

1. Определите, какие участники нужны для разработки программного продукта.
2. Оцените необходимое количество участников рабочих групп проекта, аргументируя свою точку зрения.
3. Опишите конкретные задачи каждой рабочей группы.

Определите, кого вы можете поставить в качестве лидера каждой группы, и какими профессиональными знаниями и качествами он должен обладать.

Лабораторная работа №17. Составление рабочего проекта

Теоретический материал.

Рабочий проект при создании автоматизированных систем

ГОСТ 34.601-90 «Информационная технология. Комплекс стандартов на автоматизированные системы. Автоматизированные системы. Стадии создания» определяет рабочий проект как стадию процесса создания автоматизированной системы. На этой стадии выделены два этапа работ:

- разработка предварительных проектных решений по выбранной концепции системы;
- разработка документации на автоматизированную систему и её части.

На этапе разработки предварительных решений должны быть определены следующие параметры: полный перечень функций системы, их цели и ожидаемые результаты, состав выполняемых задач, представление и структура информационной базы, функции системы управления базой данных, состав вычислительной системы – аппаратная и программная конфигурация, их функции и параметры.

Разрабатываемая на стадии рабочего проекта документация на автоматизированную систему должна содержать полную информацию о выработанных решениях. Виды и комплектность документов регламентированы ГОСТ 34.201-89.

Общесистемные решения отражаются в пояснительной записке к рабочему проекту, в схемах организационной и функциональной структур системы. Решения по техническому обеспечению должны быть показаны в схеме структурной комплекса технических средств. На данной стадии допускается все схемы включать в состав пояснительной записки. Перечень всех документов, разработанных в рамках рабочего проекта, должен быть указан в ведомости проекта.

Также на этой стадии проектирования автоматизированной системы при необходимости может быть разработан перечень заданий на разработку специализированных (новых) технических средств. Сами технические задания разрабатываются отдельно и в состав рабочего проекта не входят.

Требования к содержанию документов приведены в методических указаниях РД50-34.698-90 «Информационная технология. Комплекс стандартов на автоматизированные системы. Автоматизированные системы. Требования к содержанию документов.

Пример рабочего проекта

УТВЕРЖДАЮ

Руководитель (заказчика ИС)

Личная подпись_Расшифровка подписи

Печать

Дата «_»_ 2004 г.

УТВЕРЖДАЮ

Руководитель (разработчика ИС)

Личная подпись_Расшифровка подписи_

Печать

Дата «_»_ 2004 г.

Рабочий проект на создание информационной системы

Система Управления Базой Данных

(наименование вида И С)

БИБЛИОТЕЧНЫЙ ФОНД РОССИЙСКОЙ ФЕДЕРАЦИИ

(наименование объекта информатизации)

СУБД «Библиотека»

(сокращенное наименование ИС)

На 8 листах

2004 г.

Действует с «_»

Содержание

Ведомость рабочего проекта

На предыдущих стадиях разработки СУБД «Пенсионный Фонд» были составлены и утверждены следующие документы:

• Техническое задание на создание информационной системы СУБД «Пенсионный Фонд», разработанное на основании ГОСТ 34.602—89 на написание ТЗ на автоматизированные системы управления от 01.01.1990 г.

Пояснительная записка к рабочему проекту

Общие положения

Данный документ является эскизным проектом на создание Системы Управления Базой Данных для Библиотечного Фонда Российской Федерации (СУБД «Библиотека»).

Перечень организаций, участвующих в разработке системы, сроки и стадии разработки, а также ее цели и назначение указаны в техническом задании на создание информационной системы.

Основные технические решения

Решения по структуре системы

СУБД «Библиотека» будет представлять собой персональную систему управления локальной базой данных, работающей на одном компьютере.

Система будет управлять реляционной базой данных, представляющей собой набор связанных между собой таблиц в формате Рагабох, доступ к которым осуществляется с помощью ключей или индексов. Сведения в одной таблице могут отражать сведения из другой, и при изменении сведений в первой таблице эти изменения немедленно отображаются во второй. Таким образом будет достигнута непротиворечивость данных.

Общая структура базы данных:

- • Анкеты организации, которые зарегистрированы в данном ПФ:
  - — Тип предприятия (Российская организация, Физическое лицо, Иностранная организация, Обособленное подразделение).
  - — Вид предприятия (Адвокаты, Бюджетное, Единый налог 6 %, Единый налог 15 %, Сельхозпродукция, Службы занятости, Фермерское хозяйство, Прочее).
  - — Регистрационный номер работодателя в ПФР (3 — 3 — 6).
  - — Свидетельство: серия, номер.
  - — Дата выдачи свидетельства (число_месяц_год).
  - — ИНН.
  - — КПП.

    - — Наименование.
    - — Юридический адрес:
- • Почтовый индекс.
- • Регион.
- • Район.
- • Город.
- • Населенный пункт.
- • Улица.
- • Дом.
- • Корпус.
- • Квартира.
- — Адрес постоянно действующего органа (при отличии от юридического).
- • Анкеты сотрудников этих организаций:
    - — Фамилия.
    - — Имя.
    - — Отчество.
    - — Пол (М/Ж).
    - — Дата рождения (Дата).
    - — Страховой номер.
    - — Место рождения (Страна, Регион, Район, Город, Населенный пункт).
    - — Гражданство.
    - — Адрес регистрации (Страна, Почтовый индекс, Регион, Район, Город, Населенный пункт, Улица, Дом, Корпус, Квартира).
    - — Адрес места жительства фактический (Страна, Почтовый индекс, Регион, Район, Город, Населенный пункт, Улица, Дом, Корпус, Квартира).
    - — Телефон домашний.
    - — Телефон служебный.
    - — Документ (Удостовер. личность).
    - — Дата выдачи (Дата).
    - — Кем выдан ().
    - — Дата заполнения (Дата).
    - — ИНН.
- • Сведения о стаже сотрудников этих организаций:
- — Страховой номер.
- — Фамилия.
- — Имя.
- — Отчество.
- — Дата рождения.
- — Территориальные условия проживания на ....
- — Таблица периодов работы со следующей структурой:
- • Начало периода (дата).
- • Конец периода (дата).
- • Вид деятельности (работа, служба соцстрах, уход-дети, безр, реабилит, уход-инвд, профзаб, пересмотр).
- • Наименование организации.
- • Должность.
- • Территориальные условия.

Решения по режимам функционирования, работы системы

СУБД «Библиотека» будет функционировать в однопользовательском режиме, а также будет способна:

- • просматривать записи базы данных (в том числе и при помощи фильтров);
- • добавлять новые записи;

- • удалять записи;
- • при входе в систему будет запрашиваться пароль.

Решения по численности, квалификации и функциям персонала АС

Указанные решения должны удовлетворять требованиям, приведенным в техническом задании на разработку системы.

Состав функций комплексов задач, реализуемых системой

Автоматизированная система должна выполнять следующие функции:

- • сделать запись о пенсионном удостоверении;
- • удалить информацию о пенсионном удостоверении;
- • выдать справку о всех пенсионных удостоверениях;
- • зарегистрировать новое предприятие в ПФ РФ;
- • удалить предприятие из базы данных;
- • выдать справку обо всех предприятиях, зарегистрированных в ПФ РФ;
- • подсчитать пенсию для работников предприятий на основании стажа;
- • выдать справку о пенсионных накоплениях работника.

Решения по составу программных средств, языкам деятельности, алгоритмам процедур и операций и методам их реализации

Для реализации АС будет использоваться среда программирования Boland Delphi 7.0 и язык программирования Object Pascal.

Для подсчета пенсии будет использоваться следующий алгоритм.

Вначале определяется стажевый коэффициент пенсионера. Он полагается равным 0,55 за общий трудовой стаж до текущей даты не менее 25 лет мужчинам и 20 лет женщинам. За каждый полный год стажа сверх указанного стажевый коэффициент увеличивается на 0,01, но не более чем на 0,20.

Затем определяется отношение заработка пенсионера к среднемесячной заработной плате в стране. Этот заработок может быть взят за этот отсчетный период или за любые 60 месяцев работы подряд, или тот, из которого была исчислена пенсия на момент реформы. Среднемесячная зарплата в стране берется за тот же самый период.

Отношение заработков учитывается в размере не свыше 1,2. Для пенсионеров, проживающих на Крайнем Севере, учитываемое соотношение выше: от 1,4 до 1,9 в зависимости от установленного в централизованном порядке районного коэффициента к зарплате.

Затем стажевый коэффициент умножается на соотношение заработков и на 1671 руб. — утвержденную для расчетов среднемесячную зарплату в стране за 111 квартал 2001 г. Это и будет пересчитанный размер трудовой пенсии по новому законодательству в обычном случае. Если он оказался менее 660 руб., то размер пенсии «доводится» до этого гарантированного минимума.

Если пенсионер является инвалидом I группы или достиг к 1 января 2002 г. возраста 80 лет и более, рассчитанный в этом порядке размер пенсии по старости увеличивается на 450 руб.

Если у пенсионера имеются лица, находящиеся на его иждивении, то рассчитанный размер пенсии увеличивается на 150 руб. на каждого иждивенца, но не более чем на трех в общей сложности.

Источники разработки

Данный документ разрабатывался на основании ГОСТ 34.698—90 на написание ТЗ на автоматизированные системы управления от 01.01.1992 г.

Приложения

СОСТАВИЛИ

Должность исполнителя_

Фамилия, имя, отчество_

Подпись_

Дата «_»_ 2007 г.

Должность исполнителя_

Фамилия, имя, отчество_

Подпись_

Дата «_»_ 2007 г.

Должность исполнителя_

Фамилия, имя, отчество_

Подпись_

Дата «_»_ 2007 г.

Практическое задание. Используя предметные области (см. список ниже) и ваши предыдущие наработки, выполните следующие действия:

1. Изучите PDF-приложения к лабораторной работе.
2. Выберите пункты, которые будут нужны для вашего проекта.
3. Составьте рабочий проект.

Лабораторная работа №19. Выбор основного и дополнительного инструментария программирования

Теоретический материал.

C# — один из наиболее популярных языков программирования в мире, хотя его начали разрабатывать еще в прошлом веке. Он задумывался как альтернатива Java, но нашел собственный, вполне успешный путь. C# преподают в большинстве технологических вузов мира. Windows — все еще самая популярная компьютерная ОС, так что выбор удобной среды разработки — актуальный вопрос.

Visual Studio

Описание: самая «правильная» среда разработки. С Visual Studio многие начинают знакомиться с языком и не расстаются с ней на протяжении всей карьеры программиста.

Плюсы:

- Официальная. Так как и язык, и среда разработки созданы в Microsoft, логично предположить, что ничего более функционального вы не найдете во всем Интернете. В некоторых случаях без Visual Studio не обойтись — например, при использовании технологий UWP и WPF.
- Бесплатная. Версии «Community edition» для рядового пользователя будет достаточно. Тем более, теперь можно подключать плагины (в отличие от старой версии Express).
- Функциональная. В Visual Studio множество качественных плагинов. С их помощью можно расширить функциональность приложения и подключить другие языки.
- Поддерживает платформы .NET. Visual Studio имеет широкие возможности по разработке приложений под Windows, в том числе в .NET-сегменте.
- Облачные хранилища. Зарегистрируйтесь в сообществе Visual Studio — и получите доступ к облачному хранилищу, где сможете располагать файлы проектов.
- Корпоративность. Технология бэклога позволяет членам команды взаимодействовать при гибкой методологии разработки.

Минусы:

- Баги при переходах с триал-версии. При переходе на платную версию могут теряться настройки и нарушаться работа корпоративного сервера.
- Сложность. Самостоятельно освоить Visual Studio новичку будет непросто — слишком много доступных функций, спрятанных в подразделах меню.

Project Rider

Описание: среда от JetBrains для работы с платформой .NET. Выпущена в прошлом году, но уже приобрела много поклонников.

Плюсы:

- ReSharper. Это плагин, изначально разработанный для повышения производительности Visual Studio. Теперь на его основе выпущена IDE.
- Поддержка полного цикла. Фирменная черта продуктов JetBrains, воплощенная и в Project Rider. С ним вы сможете организовать весь цикл создания ПО: от идеи до поддержки.
- Функциональность. Project Rider позволяет подключить MSBuild и XBuild, работать с CLI-проектами и организовать отладку приложений .NET and Mono. Множество опций для быстрого создания кода улучшает производительность.
- Multiple runtime. Поддержка нескольких запущенных программ.
- Кроссплатформенность. Project Rider работает с Windows, Linux и MacOS.
- Контроль версий. Встроенный инструмент позволяет напрямую организовать работу с Git, Mercurial и TFS.

Минусы:

- Молодость. Часть функциональности еще в разработке, не все стартовые баги исправлены.

- Стоимость. Самая дешевая версия Project Rider обойдется в 139 долларов за первый год использования. Но есть триал-версия и специальные предложения для студентов и непрофильных организаций.

Eclipse

Описание: одна из самых популярных мультиязычных сред. Ориентирована преимущественно на разработку Java-приложений, но полезна и для кодов на C#.

Плюсы:

- Множество плагинов. У Eclipse едва ли не самое большое число надстроек — «на все случаи жизни».
- Активное сообщество. Помогает быстрее освоить среду разработки, выпускает новые плагины.
- Отличные компилятор и отладчик. Первый работает на порядок быстрее, чем у конкурентов, второй — показывает потоки, пересечения, позволяет гибко управлять ходом отладки.
- Кастомизация. Благодаря плагинам и настройкам можно полностью персонализировать Eclipse.
- Бесплатность. Это open-source проект, абсолютно бесплатный.
- Высокая функциональность. Благодаря разработчикам-официалам и членам сообщества с помощью Eclipse можно провести любой C#-продукт по полному циклу разработки.

Минусы:

- Сложность. Как и любой функциональный продукт, Eclipse может показаться новичку слишком сложным.
- Нет гарантий надежности. Так как плагины создаются сообществом, за их качество отвечает только разработчик. Кроме того, сами создатели Eclipse с каждой новой версией плодят баги, не успевая порой исправлять старые.

Visual Studio Code

Описание: кроссплатформенный редактор кода, который при помощи плагинов можно «подтянуть» к статусу IDE.

Плюсы:

- Кроссплатформенность. Работает на MacOS, Ubuntu и Windows. Пока недоступен на Android и iOS.
- Бесплатность. Простой open-source редактор и плагины — платить не надо.
- Легковесность. Потребуется совсем мало ресурсов, чтобы приступить к работе с минималистичным VSC.

Минусы:

- Низкая функциональность. Несмотря на поддержку .NET-платформы, VCS неудобен для сложных проектов.
- Сомнительная надежность. Многие надстройки имеют низкое качество сборки и не всегда выполняют даже основные функции.

MonoDevelop

Описание: свободная среда разработки от Xamarin для создания приложений на множестве языков, в том числе на C#.

Плюсы:

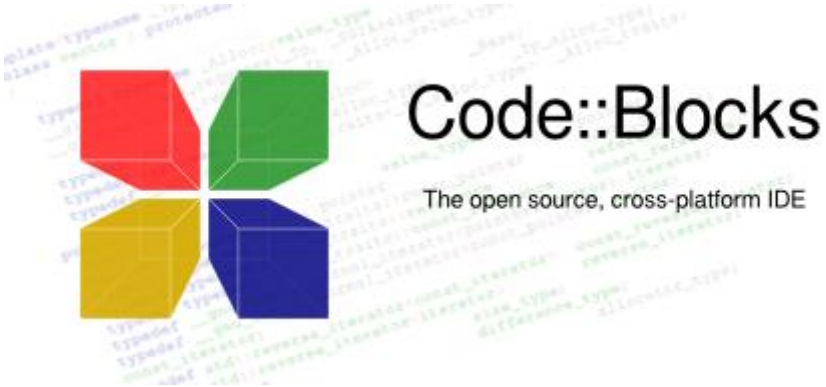
- Мультиплатформенность. Поддерживает Linux, Windows и Mac OS.
- Кастомизация. На рабочем столе можно расположить функции и окна по своему усмотрению.
- Unity 3D. Полноценная поддержка популярной платформы для разработки игр.

- Бесплатность.

  Минусы:

- Ограниченная функциональность. У MonoDevelop есть собственный отладчик и инструменты для работы с кодом. Но в части поддержки разных платформ и проектов — это пока сырая IDE.

Code::Blocks



Описание: среда разработки, известная простой и удобством в настройке и использовании.

Плюсы:

- Бесплатность. Полноценный open-source проект.
- Простота. В отличие от Visual Studio, среда Code::Blocks понятна новичку, знающему один из поддерживаемых языков.
- Кроссплатформенность. IDE запускается на любой десктопной ОС.
- Выбор компилятора. Code::Blocks ограничена в функциональности, но эта возможность — несомненный плюс.
- Легковесность.

  Минусы:

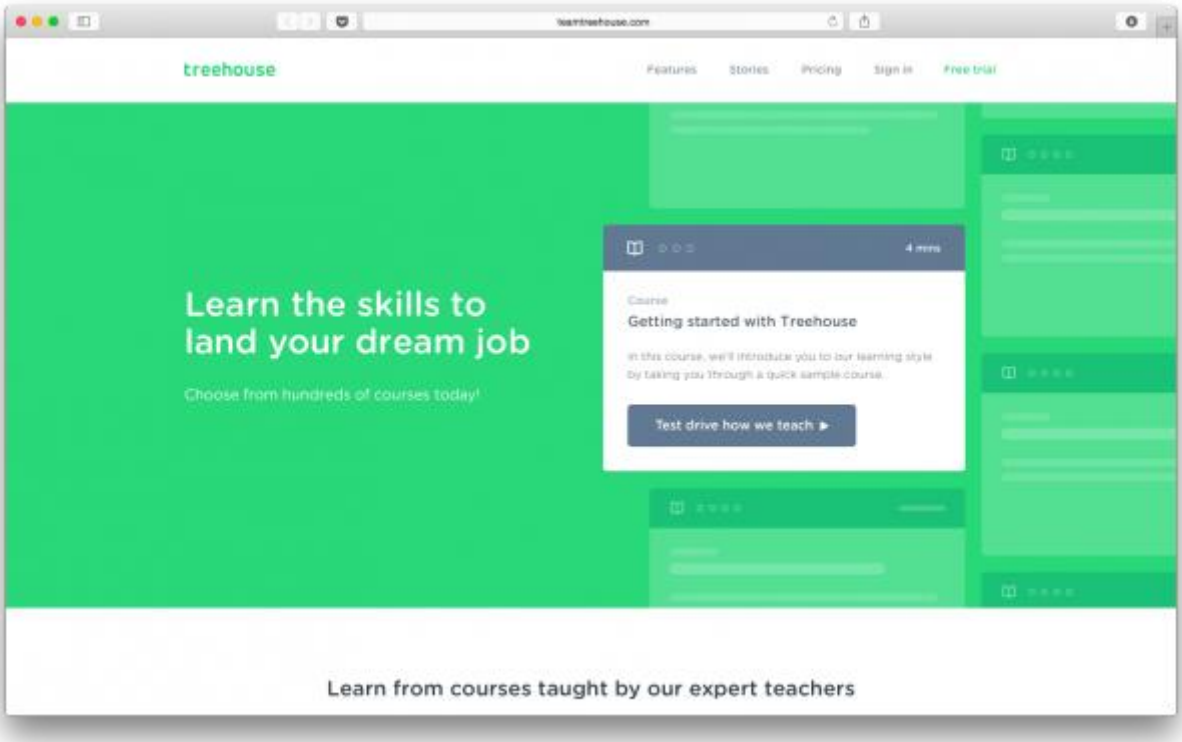
- Недостаточная функциональность. Для создания комплексных приложений Code::Blocks категорически не подходит.
- Нестабильность. Приходится сталкиваться с нелепыми ошибками в отладке и некорректной работой всей среды.

Заключение

Список IDE, получаемых даже при подключении сторонних плагинов, мал относительно Java или PHP. Но это тот случай, когда малое количество компенсируется качеством: в числе предложенных сред каждый сможет найти подходящую — по планируемым задачам и потребляемым ресурсам.

Инструменты разделены на разные категории: платформы для разработки, обучение программированию, багтрекинг, API и прочее. Не все инструменты бесплатные, но за удобство и новые функции приходится платить. Надеемся, вы найдёте что-то полезное для себя.

Обучение программированию



Treehouse

1. Treehouse — обучение дизайну и разработке для веб и iOS.
2. Codeacademy — интерактивный и весёлый способ научиться программированию.
3. Code School — практические курсы для веб-разработчиков.
4. Udacity — обучение посредством решения практических задач от известных преподавателей.
5. Coursera — множество компьютерных курсов, причём бесплатных.
6. RubyMonk — интерактивные курсы по Ruby.
7. Khan Academy — бесплатное образование с огромным множеством курсов по программированию.
8. School of Webcraft — наполняемый пользователями ресурс по веб-разработке.
9. Google Code University — гайды, курсы и обучающие материалы от Google.
10. Orientation to Android Training — официальный курс по разработке на Android.
11. Phpacademy — бесплатные видеоуроки по PHP.
12. Hexlet — обучение программированию в реальной среде разработки.

Системы контроля версий

GitHub

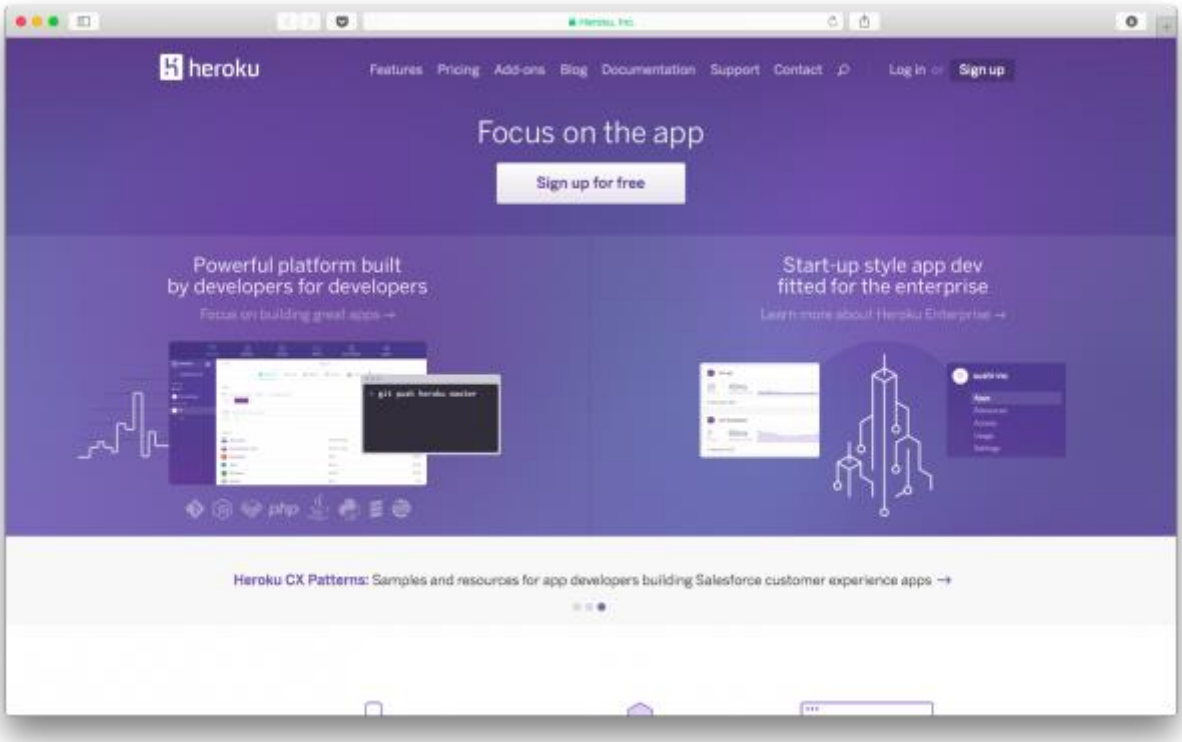
1. GitHub — хостинг для IT-проектов.
2. Pixelapse — сервис, который показывает, как выглядел код в прошлых версиях.
3. Bitbucket — бесплатный хостинг для кода.
4. Versions — Mac-клиент для сервиса Subversion.
5. SourceTree — бесплатный Mac-клиент для систем Git и Mercurial.
6. OFFSCALE — управление версиями баз данных.
7. Tower — Git-клиент для Mac.

Разное

AppNeta

1. AppNeta — облачная APM (Application Performance Management).
2. TaskMissile — сервис для получения фидбэка от пользователей.
3. Kera — создание встроенных в приложение туториалов для пользователей.
4. Flowdock — почтовый клиент и чат для команд.
5. Modulus — хостинг для Node.js и MongoDB.
6. Metricfire — сервис для отслеживания различных метрик.
7. Interstate — позволяет превратить обычных пользователей в лояльных.
8. Codenow — совместная работа для программистов. Легко находить код и делиться им.
9. Lingohub — локализация софта.
10. TranslateKarate — простой онлайн-сервис для перевода и локализации.
11. Kickfolio — тестирование, поддержка, маркетинг и реклама. Всё в одном.
12. Snippets — сервис для хранения сниппетов.
13. Product Hunt — множество идей для новых приложений.

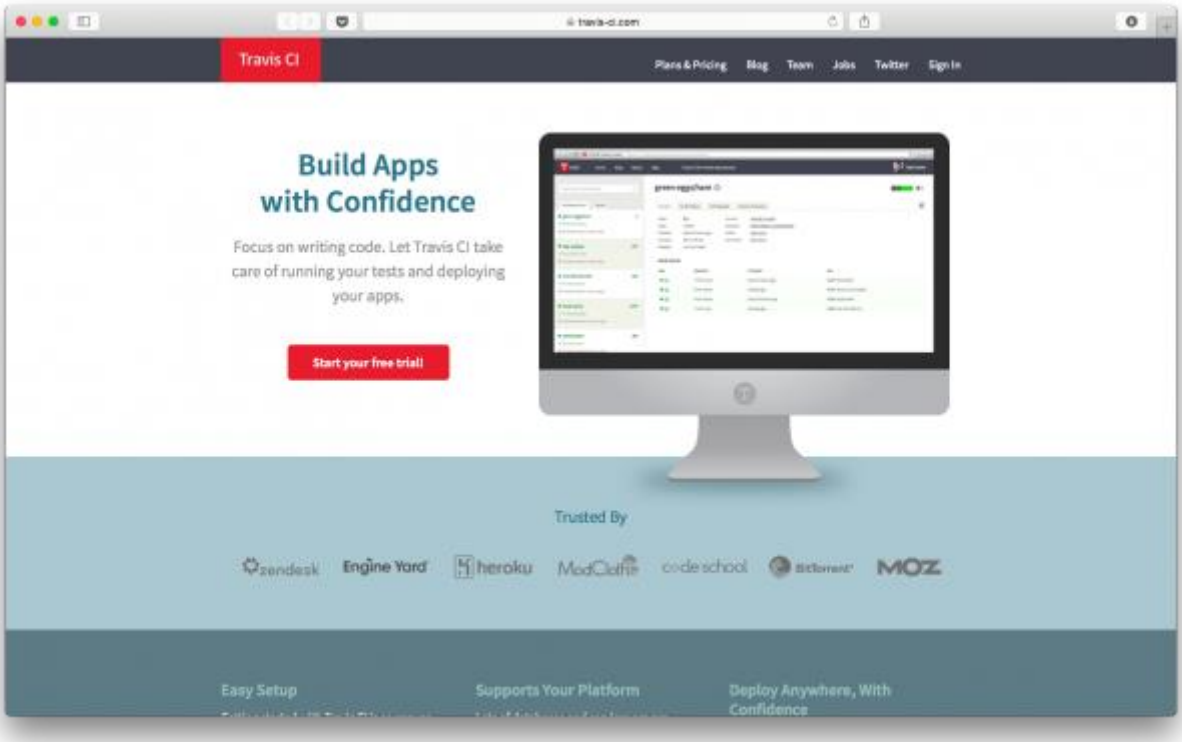
Платформы для разработки



Heroku

1. Heroku — облачная платформа для создания приложений.
2. Compilr — позволяет следить за кодом с любого браузера.
3. Kinvey — облачный back-end для мобильных приложений.
4. Firebase — back-end для сайта.
5. Cloud9 — IDE онлайн.
6. Parse — полноценная платформа для мобильных приложений.
7. CloudMine — back-end для мобильных и веб-приложений.
8. Koding — IDE в браузере. Новый способ работы для разработчиков.
9. AppHarbor — облачная платформа .NET.
10. dotCloud — помощь в разработке и расширении веб-приложений.
11. BrainEngine — облачная платформа для Force.com.
12. PHP Fog — облачная платформа для PHP.
13. Backrest — лёгкое создание back-end для SaaS.
14. Codeanywhere — онлайн-редактор кода.
15. NeptuneIDE — облачная IDE для PHP.
16. Fusegrid — облако для ColdFusion.
17. Cloud IDE — написание кода и дебаггинг в онлайне.
18. FriendCode — социальная сеть для программистов.
19. ToolsCloud — среда для разработки в онлайне.

Интеграция и развёртывание



Travis CI

1. Travis CI — интеграция и развёртывание для мобильных приложений.
2. CircleCi — интеграция и развёртывание для веб-приложений.
3. Railsonfire — интеграция и развёртывание для софта на Ruby.
4. Wercker — платформа для создания и интеграции приложений.
5. Hostedci — интеграция и развёртывание для приложений на iOS и OS X.

Обратная связь, мониторинг и багтрекинг

Crashalytics

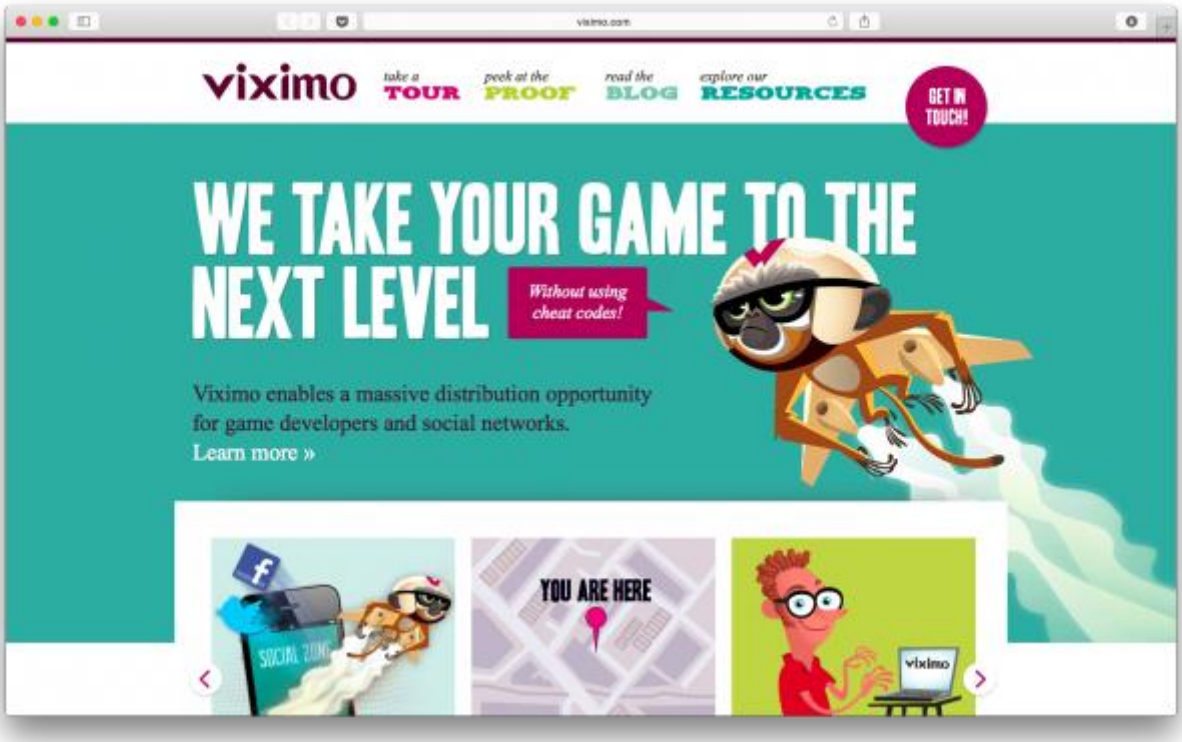
1. Crashlytics — система для отслеживания крэшей приложений на iOS и Android.
2. Usersnap — делает скриншот багов в приложениях.
3. Crittercism — платформа для мониторинга производительности.
4. Rollbar — отчёт и трекинг багов в реальном времени.
5. New Relic — APM для веб-приложений.
6. Exceptional — отслеживание ошибок в веб-приложениях в реальном времени.
7. BugSense — система для слежения за крэшами в мобильных приложениях.
8. Bugzilla — серверное ПО для управления разработкой приложения.
9. Bugify — отслеживание ошибок в PHP-коде для небольших команд.
10. BugHerd — простой багтрекер.
11. Snowy Evening — отслеживание багов и интеграция с GitHub.

API

Twilio

1. Twilio — API для мессенджеров и VoIP.
2. OpenWeatherMap — бесплатные API погоды.
3. Stripe — платёжная система для разработчиков.
4. Factual — API структурированной информации.
5. Filepicker.io — упрощение работы с контентом, созданным пользователями (UGC).
6. PubNub — сервис для обмена сообщениями в реальном времени в облаке.
7. Mailgun — почта для разработчиков.
8. Context.IO — API для почтовых клиентов.
9. Semantics3 — API для информации о товарах.
10. Redpin — система для навигации в помещении.
11. Sent.ly — API для SMS-общения с пользователями.
12. Embedly — конвертирование URL в видео, фото и другое.

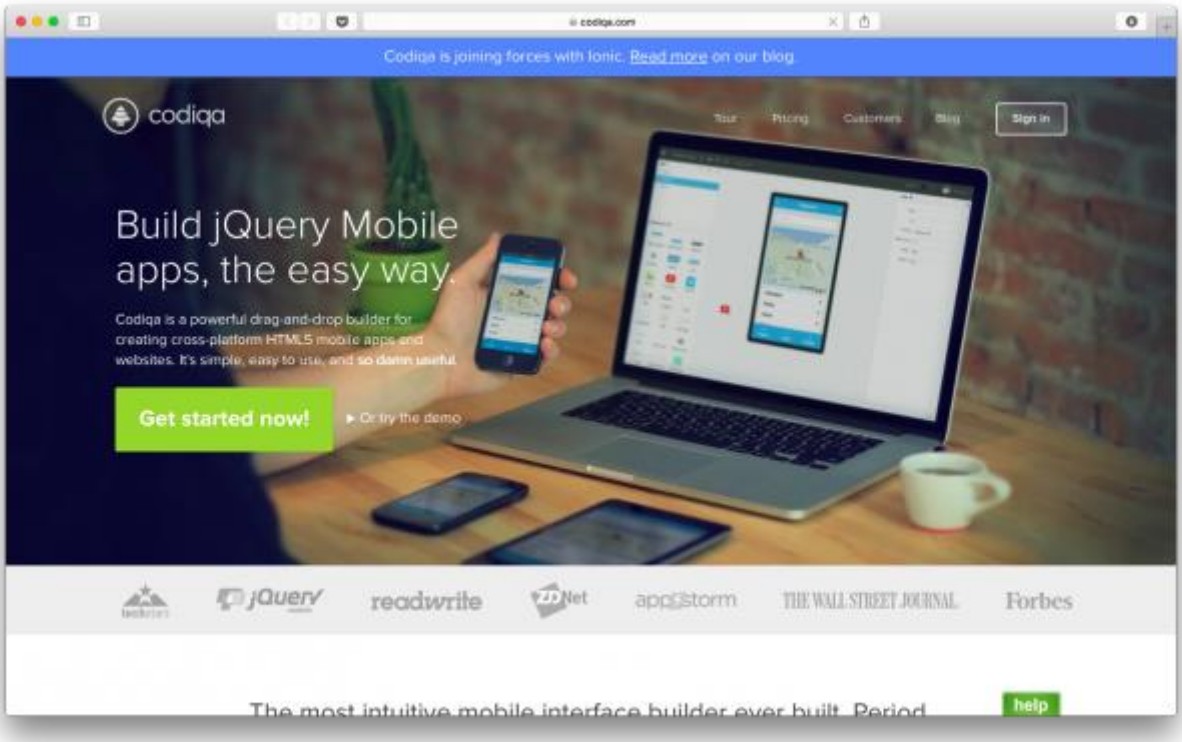
Разработка игр



Viximo

1. Viximo — платформа для дистрибуции социальных игр.
2. XNA — инструменты для разработки игр от Microsoft.
3. Yodo1 — платформа для дистрибуции игр в Китае.
4. Game Closure — SDK для игр на JavaScript.
5. FTW — синхронизация сохранений, счёта и друзей между устройствами.
6. Storybricks — создание собственной MMO.

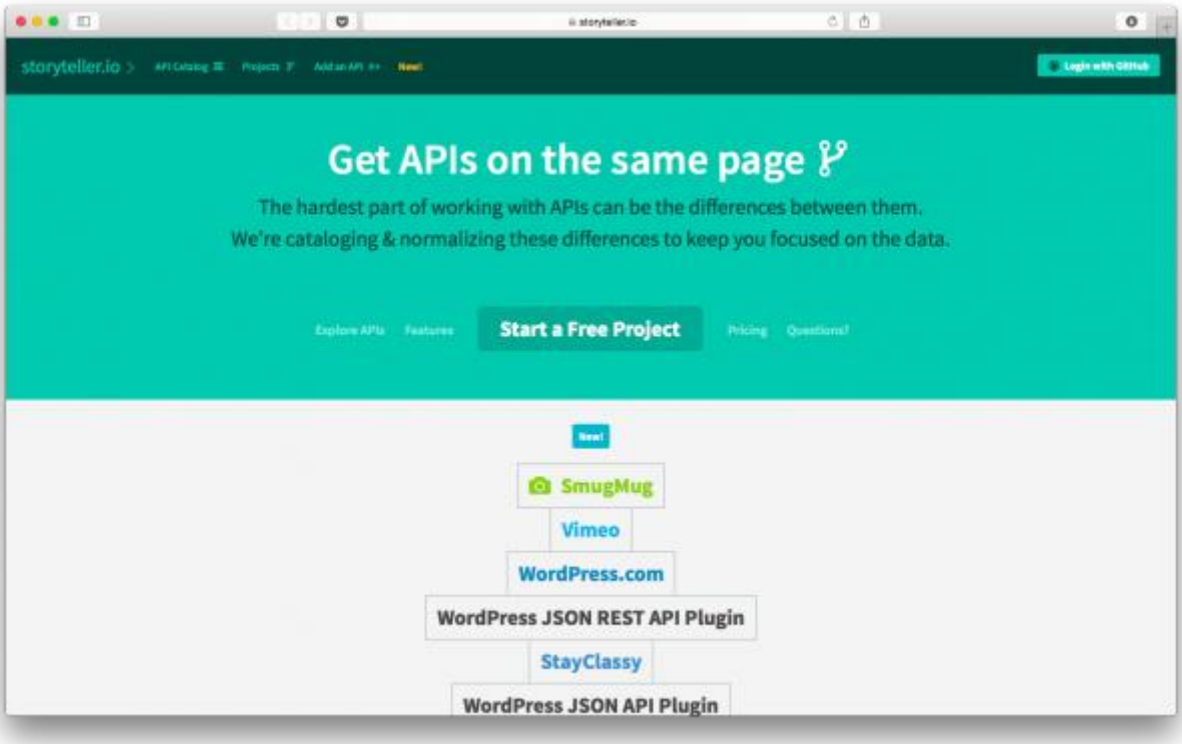
Разработка мобильных приложений



Codiqa

1. Codiqa — быстрое создание прототипа мобильного приложения.
2. AppCooker — генератор мокапов для мобильных приложений.
3. Apptentive — обратная связь для мобильных приложений.
4. AppCod.es — SEO и маркетинг в App Store.
5. Chupa Mobile — рынок для компонентов мобильных приложений.
6. Appboy — аналитика, CRM и прочее.
7. Flurry — аналитика, трафик и монетизация.
8. Octopod — платформа для разработки мобильных приложений.
9. Little Eye — слежение за потреблением батареи для приложений на Android.

Вне категории



Storyteller

1. Binpress — рынок для покупки скриптов и компонентов для разработки.
2. UploadCare — сервис для загрузки и хранения кода.
3. Eden — библиотека PHP для быстрого прототипирования.
4. Appbackr — краудфандинговая платформа для мобильных приложений.
5. Modkit — программирование для чего угодно.
6. TechScratch — сервис помогает сфокусироваться на том, что вы делаете лучше всего, и помогает со всем остальным.
7. Storyteller — создание контентных сайтов.
8. Feed.Us — CMS для веб-приложений.
9. Hosted Graphite — информация в виде понятных графиков и диаграмм.
10. Divshot — создание интерфейсов для веб-приложений. Быстрое прототипирование на HTML 5.
11. FlyWithMonkey — инструменты для разработчиков на HTML5.
12. Expanz — помогает с разработкой приложений для бизнеса.
13. Zapstreak — AirPlay для Android.
14. RepoDrop — приватный репозиторий для кода.
15. CodeWars — тренировка и проверка своей способности к программированию.
16. Architexa — помогает понять сложные части кода в Java.
17. UserMetrics — аналитика того, как пользователи используют ваше приложение.
18. Setapp — находите и делитесь полезными инструментами.

19. Coder Bounty — устанавливайте награду за решение проблем в коде.
20. Last5 — отслеживание времени и продуктивности для разработчиков.
21. XtGem — система для создания сайтов.
22. uTest — тестирование приложений.

Практическое задание. Используя предметные области (см. список ниже), выполните следующие действия:

1. Опишите все преимущества Visual Studio Community над остальными средами разработки.
2. Выберите инструменты для тестирования разработанного ПО.
3. Выберите инструменты для создания диаграмм SADT и UML.
4. Определите инструментарий для создания документации, включая руководства пользователей различного уровня.

Примечание: выбор необходимо доходчиво аргументировать.

Лабораторная работа №20. Разработка ПП: создание базы данных.

Теоретический материал.

Проектирование базы данных – это процесс создания проекта базы данных, предназначенной для поддержки функционирования экономического объекта и способствующей достижению его целей. Оно представляет собой трудоемкий процесс, требующий совместных усилий аналитиков, проектировщиков и пользователей. При проектировании базы данных необходимо учитывать тот факт, что база данных должна удовлетворять комплексу требований.

Эти требования следующие.

1. Целостность базы данных. (Требование полноты и непротиворечивости данных).

2. Многократное использование данных.

3. Быстрый поиск и получение информации по запросам пользователей.

4. Простота обновления данных.

5. Уменьшение излишней избыточности данных.

6. Защита данных от несанкционированного доступа, от искажения и уничтожения.

Жизненный цикл базы данных (ЖЦБД) – это процесс проектирования, реализации и поддержки базы данных. ЖЦБД состоит из следующих семи этапов:

1) предварительное планирование;

2) проверка осуществимости;

3) определение требований;

4) концептуальное проектирование;

5) логическое проектирование;

6) физическое проектирование;

7) оценка работы и поддержка базы данных.

Опишем главные задачи каждого этапа.

1. Предварительное планирование базы данных. Это важный этап в процессе перехода от разрозненных к интегрированным данным. На этом этапе собирается информация об используемых и находящихся в процессе разработки прикладных программах и файлах, связанных с ними. Она помогает установить связи между текущими приложениями и то, как используется информация приложений. Кроме того, позволяет определить будущие требования к базе данных.

2. Проверка осуществимости. Она предполагает подготовку отчетов по трем вопросам:

1) есть ли технология – необходимое оборудование и программное обеспечение – для реализации запланированной базы данных? (*технологическая осуществимость*);

2) имеются ли персонал, средства и эксперты для успешного осуществления плана создания базы данных? (*операционная осуществимость*);

3) окупится ли запланированная база данных? (*экономическая эффективность*).

3. Определение требований. На этом этапе определяются:

· цели базы данных;

· информационные потребности различных структурных подразделений и их руководителей;

· требования к оборудованию;

· требования к программному обеспечению.

4.Концептуальное проектирование. На этом этапе создаются подробные модели пользовательских представлений данных предметной области. Затем они интегрируются в *концептуальную модель*, которая фиксирует все элементы корпоративных данных, подлежащих загрузке в базу данных. Эту модель называют еще *концептуальной схемой базы данных.*

5.Логическое проектирование. На этом этапе осуществляется выбор типа модели данных. Концептуальная модель отображается в *логическую модель*, основанную уже на структурах, характерных для выбранной модели. Так, если выбрана реляционная модель, то разрабатываются структуры таблиц, определяются их ключи, устанавливается связь между таблицами, оптимизируется созданная модель базы данных (минимизируется избыточность данных). Наиболее распространенным методом при оптимизации является метод нормальных форм или, другими словами, *нормализация данных*

6.Физическое проектирование. На этом этапе предусматривается принятие разработчиком окончательного решения о способах реализации создаваемой базы данных. Логическая модель расширяется характеристиками, необходимыми для определения способов физического хранения базы данных, типа устройств для хранения, методов доступа к данным базы, требуемого объема памяти, правил сопровождения базы данных и др.

7.Оценка и поддержка базы данных. Оценка включает опрос пользователей на предмет выяснения, какие их информационные потребности остались неучтенными. При необходимости в спроектированную базу данных вносятся изменения. Пользователи обучаются работе с базой данных. По мере расширения и изменения потребностей бизнеса поддержка базы данных обеспечивается путем внесения изменений, добавления новых данных, разработки новых прикладных программ, работающих с базой данных.

Практическое задание. Используя предметные области (см. список ниже), выполните следующие действия:

1. Используя предыдущие наработки и диаграммы IDEF1X, постройте схему данных вашей БД.
2. Реализуйте базу данных в среде Microsoft SQL Server.

Лабораторная работа №21. Разработка ПП: определение дизайна и

визуализация.

Теоретический материал.

Шаблоны проектов и элементов Visual Studio для приложений для Windows

Visual Studio 2019 предоставляет множество шаблонов проектов и элементов. Такие шаблоны помогают создавать приложения для устройств с Windows 10 на C# или C++. В этом разделе описаны шаблоны и приведены рекомендации по выбору одного из них для вашего сценария.

- К шаблонам проектов относятся файлы проекта, файлы кода и другие ресурсы, настроенные для создания приложения или компонента, которые может загрузить и использовать приложение.
- Шаблоны элементов — это файлы проекта, которые содержат часто используемые код и XAML, которые можно добавить в проект, чтобы ускорить разработку. Например, с помощью шаблона элемента можно добавить в приложение новое окно, страницу или элемент управления.

Шаблоны WinUI

Библиотека пользовательского интерфейса Windows (WinUI) — это современная нативная платформа пользовательского интерфейса, поддерживаемая приложениями для Windows, как классическими (.NET и нативные приложения Win32), так и приложениями UWP. WinUI 3 (сейчас доступна предварительная версия для разработчиков) — это последняя основная версия WinUI. Это полноценная платформа пользовательского интерфейса для классических приложений для Windows.

WinUI 3 включает пакет VSIX для Visual Studio 2019 с шаблонами проектов и элементов, которые помогут приступить к созданию приложений с помощью интерфейса на основе WinUI. Дополнительные сведения о пакете VSIX WinUI 3 и содержащихся в нем шаблонах проектов см. в этом разделе.

Важно!

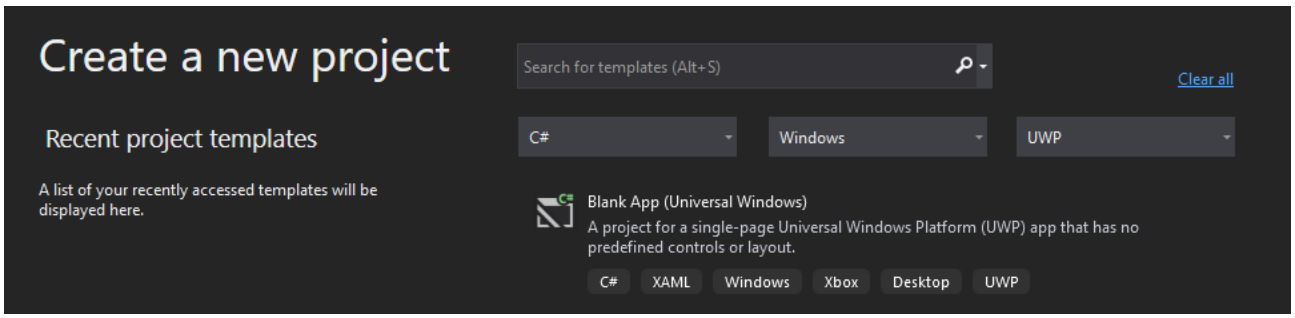
WinUI 3, включая связанные шаблоны Visual Studio, сейчас предоставляется в предварительной версии для разработчиков, предназначенной для раннего ознакомления и сбора отзывов от сообщества разработчиков. Пока ее НЕ СЛЕДУЕТ использовать для приложений в рабочей среде.

Шаблоны UWP

Visual Studio предоставляет разнообразные шаблоны проектов для создания приложений UWP с помощью C# или C++. Чтобы использовать эти шаблоны проектов, при установке Visual Studio необходимо включить рабочую нагрузку **Разработка приложений для универсальной платформы Windows**. Для шаблонов проектов C++ также необходимо включить дополнительный компонент **средств универсальной платформы Windows C++ (v142)** для рабочей нагрузки **Разработка приложений для универсальной платформы Windows**.

Шаблоны проектов для C# и UWP

Чтобы получить доступ к шаблонам проектов UWP C#, когда вы создаете проект в Visual Studio, отфильтруйте язык, выбрав **C#** , платформу, выбрав **Windows**, и тип проекта, выбрав значение **UWP**.



Эти шаблоны проектов можно использовать для создания приложений UWP на C#.

С помощью этих шаблонов проектов можно создавать фрагменты приложений UWP на C#.

Шаблоны проектов для C++ и UWP

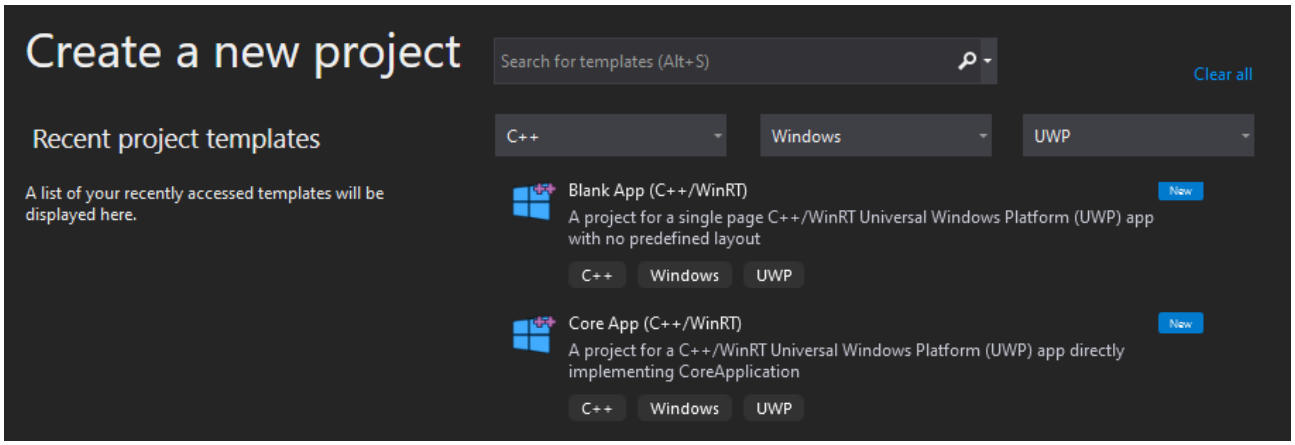
Для создания приложений UWP можно использовать две технологии C++:

- Рекомендуемая технология — C++/WinRT. Это проекция языка C++, которая полностью реализована в файлах заголовков и предназначена для предоставления эффективного и удобного доступа к современному API WinRT.
- Кроме того, можно использовать более старый набор расширений C++/CX. C++/CX по-прежнему поддерживается, но мы рекомендуем вместо него использовать C++/WinRT.

Чтобы получить доступ к шаблонам проектов UWP C++, когда вы создаете проект в Visual Studio, отфильтруйте язык, выбрав **C++** , платформу, выбрав **Windows**, и тип проекта, выбрав значение **UWP**.

Примечание

По умолчанию рабочая нагрузка **Разработка приложений для универсальной платформы Windows** в Visual Studio предоставляет доступ только к шаблонам проектов C++/CX. Чтобы получить доступ к шаблонам проектов C++/WinRT, необходимо установить пакет **VSIX C++/WinRT**.



Эти шаблоны проектов можно использовать для создания приложений UWP на C++.

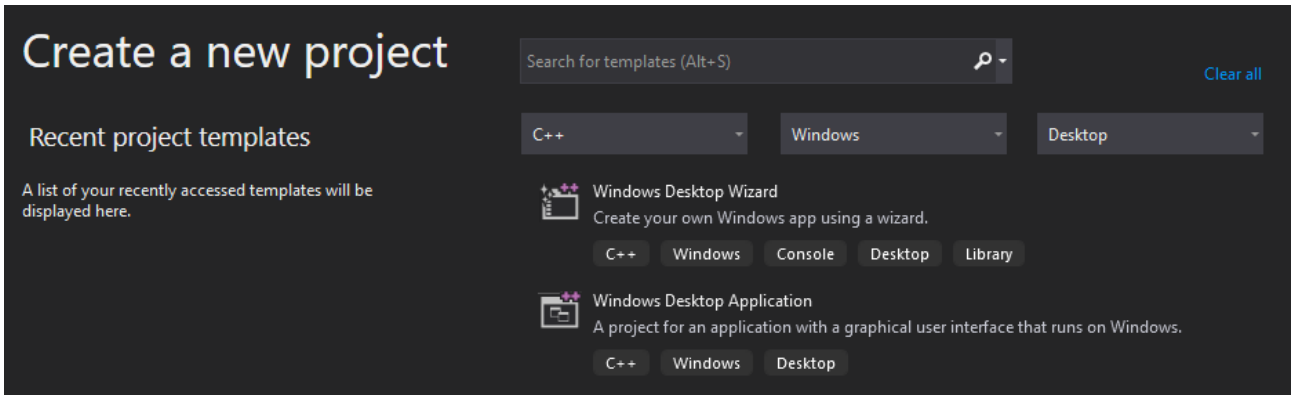
С помощью этих шаблонов проектов можно создавать фрагменты приложений UWP на C++.

Шаблоны C++/Win32

Visual Studio предоставляет разнообразные шаблоны проектов для создания классических приложений для Windows с использованием нативного C++ и прямым доступом к API Win32. Чтобы использовать эти шаблоны проектов, при установке Visual Studio необходимо включить рабочую нагрузку **Разработка классических приложений на C++** . Эта рабочая нагрузка включает шаблоны проектов для создания классических и консольных приложений, а также библиотек.

Шаблоны проектов для классических приложений

Чтобы получить доступ к шаблонам проектов C++ для классических приложений, когда вы создаете проект в Visual Studio, отфильтруйте язык, выбрав **C++** , платформу, выбрав **Windows**, и тип проекта, выбрав значение **Desktop** (Классическое приложение).

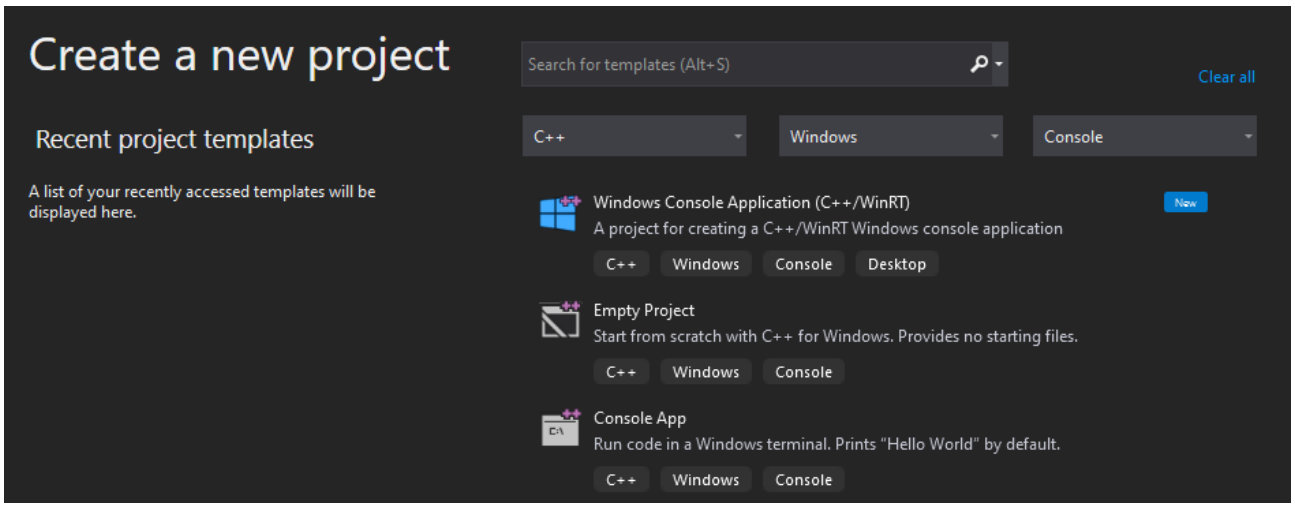


ШАБЛОНЫ ПРОЕКТОВ ДЛЯ КЛАССИЧЕСКИХ ПРИЛОЖЕНИЙ

Шаблон	Описание
Классическое приложение для Windows	Создает классическое приложение для Windows с использован C++. Дополнительные сведения см. в статье Пошаговое руководство. Созда традиционного классического приложения для Windows (C++).
Мастер классических приложений Windows	Это пошаговый мастер, с помощью которого можно создать один из следую типов проектов: классическое приложение для Windows, консольное приложе библиотека динамической компоновки (DLL) или статичес библиотека. Дополнительные сведения см. в статьях Мастер классичес приложений Windows и Пошаговое руководство. Создание традицион классического приложения для Windows (C++).
Проект упаковки приложений Windows	Создает проект, с помощью которого можно разработать классичес приложение в пакете MSIX. Благодаря этому пользователь получает современ методы развертывания, возможности интеграции с компонентами Window через расширения пакетов и многое другое. Дополнительные сведения см. в ста о Проекте упаковки приложений Windows.

Шаблоны проектов для консольных приложений

Чтобы получить доступ к шаблонам проектов C++ для консольных приложений, отфильтруйте язык, выбрав **C++** , платформу, выбрав **Windows**, и тип проекта, выбрав значение **Console** (Консольное приложение).

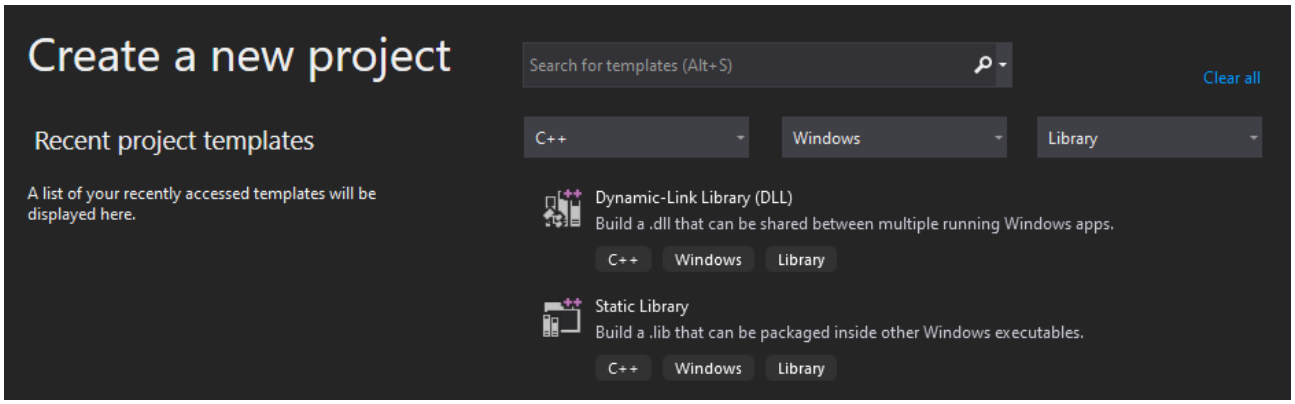


ШАБЛОНЫ ПРОЕКТОВ ДЛЯ КОНСОЛЬНЫХ ПРИЛОЖЕНИЙ

Шаблон	Описание
Консольное приложение для Windows (C++/WinRT)	Создает консольное приложение C++/WinRT без пользовательск интерфейса. Дополнительные сведения см. в статье Краткое руководство C++/WinRT. Для этого шаблона проекта требуется VSIX C++/WinRT.
Консольное приложение	Создает консольное приложение без пользовательск интерфейса. Дополнительные сведения см. в статье Пошаговое руководс Создание стандартной программы C++.
Пустой проект	Пустой проект для создания приложения, библиотеки или DLL. Необход добавить требуемый код или ресурсы.

Шаблоны проектов для библиотек

Чтобы получить доступ к шаблонам проектов C++ для библиотек, отфильтруйте язык, выбрав **C++** , платформу, выбрав **Windows**, и тип проекта, выбрав значение **Библиотека**.



ШАБЛОНЫ ПРОЕКТОВ ДЛЯ БИБЛИОТЕК

Шаблон	Описание
Библиотека динамической	Проект для создания библиотеки динамической компоно (DLL). Дополнительные сведения см. в статье Пошаговое руководс

ШАБЛОНЫ ПРОЕКТОВ ДЛЯ БИБЛИОТЕК

Шаблон	Описание
компоновки (DLL)	Создание и использование собственной библиотеки динамической компоно (C++).
Статическая библиотека	Проект для создания статической библиотеки (LIB). Дополнительные сведе см. в статье Пошаговое руководство. Создание и использование статичес библиотеки.

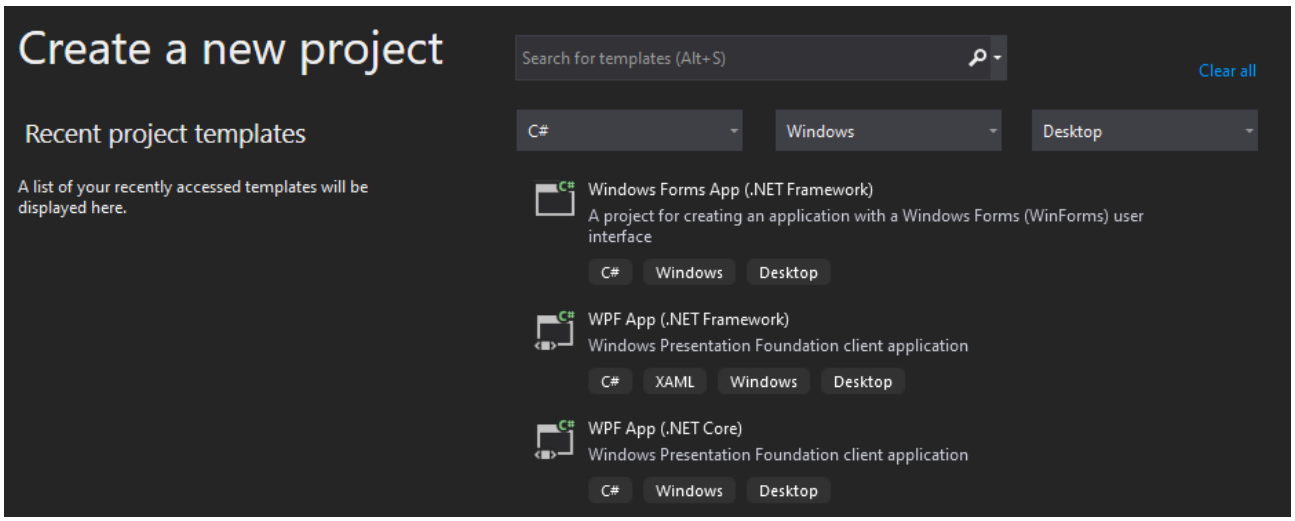
Шаблоны элементов для нативного C++ и Win32

Шаблоны проектов C++ включают множество шаблонов элементов, с помощью которых можно выполнять такие задачи, как добавление в проект новых файлов и ресурсов. Полный список см. в статье об использовании шаблонов Visual C++ в диалоговом окне "Добавление нового элемента".

Шаблоны .NET

Visual Studio предоставляет разнообразные шаблоны проектов для создания классических приложений для Windows, использующих .NET и C#. Чтобы использовать эти шаблоны проектов, при установке Visual Studio необходимо включить рабочую нагрузку **Разработка классических приложений .NET**.

Чтобы получить доступ к шаблонам проектов .NET C#, когда вы создаете проект в Visual Studio, отфильтруйте язык, выбрав **C#**, платформу, выбрав **Windows**, и тип проекта, выбрав **Desktop** (Классическое приложение).



С помощью этих шаблонов проектов можно создавать приложения с использованием C# и .NET.

ШАБЛОНЫ .N

Шаблон	Описание
Приложение WPF (.NET Core)	Создает приложение WPF, предназначенное для .NET Core. Пошаговые инструкции по этому шаблону проекта см. в статье о создании приложения WP

Шаблон	Описание
Приложение WPF (.NET Framework)	Создает приложение WPF, предназначенное для .NET Framework. Пошаговые инструкции по этому шаблону проекта см. в руководстве по созданию первого приложения WPF.
Приложение Windows Forms (.NET Core)	Создает приложение Windows Forms, предназначенное для .NET Core.
Приложение Windows Forms (.NET Framework)	Создает приложение Windows Forms, предназначенное для .NET Framework. Пошаговые инструкции по этому шаблону проекта см. в статье Создание приложения Windows Forms на C# в Visual Studio.
Проект упаковки приложений Windows	Создает проект, с помощью которого можно разработать приложение WPF или Windows Forms в пакете MSIX. Благодаря этому пользователь получает современные методы развертывания, возможности интеграции с компонентами Windows 10 через расширения пакетов и многое другое. Дополнительные сведения см. в статье о Проекте упаковки приложений Windows.

Любая форма включает в себя *заголовки, поля с подписями и подсказками, кнопки*. При успешном заполнении или о том, что что-то пошло не так, нужно сообщить, поэтому дополнительно к форме нужно спроектировать *успешную отправку* и *валидацию с сообщениями об ошибках*. Рассмотрим подробнее, о чем важно помнить в каждом компоненте.

Заголовок (title)

Форму важно назвать, не игнорируйте заголовок. В названии в 1–3 словах должна кратко фигурировать конечная цель заполнения формы. “Вход”, “Регистрация”, “Оформление заказа” и т.д.

Определите также, нужно ли интро перед формой или короткий текст рядом с заголовком, поясняющий для чего форма нужна.

Проанализируйте, каким образом пользователь попадает на форму и понимает ли он к этому моменту ценность ее заполнения. Если по пути к форме он еще не замотивирован, дайте же ему его сладкую морковку. Формулируя текст, сосредоточьтесь на ценности, которую получит пользователь, а не получатель данных.

Поля (input fields)

Это могут быть текстовые поля (text fields), поля пароля (password fields), флажки (checkboxes), радиокнопки (radio buttons), ползунки (sliders) и другие.

Минимизируйте запрос

Обязательно спрашивайте только то, что вам действительно нужно. Каждое дополнительное поле, добавленное в форму, будет влиять на ее конверсию. Всегда думайте о том, почему вы запрашиваете определенную информацию от пользователя и как вы будете ее использовать.

Рекомендации по типографике

Помните об accessibility. Шрифт должен быть достаточно крупным, чтобы текст легко читался. Безопасный вариант — 16 px для основного текста. Конечно, размер шрифта зависит от контента и от других элементов на странице, но если вы сомневаетесь — выбирайте более крупный вариант.

Практическое задание. Используя предметные области (см. список ниже), выполните следующие действия:

1. Воспользуйтесь шаблонами для создания форм в вашей программе, если в ней необходимы доработки.
2. Доработайте ранее созданные формы в вашей программе, соблюдая рекомендации, приведенные выше.

Лабораторная работа №22. Разработка ПП: программирование

Теоретический материал.

C#: требования и рекомендации по написанию кода

1. Требования

1.1 Pascal casing

Описываются имена:
• всех определений типов, в том числе пользовательских классов, перечислений, событий, делегатов и структур;
• значения перечислений;
• readonly полей и констант;
• интерфейсов;
• методов;
• пространств имен (namespace);
• свойств;
• публичных полей;

```
namespace SampleNamespace
{
  enum SampleEnum
  {
```

```
  FirstValue,
  SecondValue
 }

 struct SampleStruct
 {
  public int FirstField;
  public int SecondField;
 }

 interface ISampleInterface
 {
  void SampleMethod();
 }

 public class SampleClass: SampleInterface
 {
  const int SampleConstValue = 0xffffff;

  readonly int SampleReadonlyField;

  public int SampleProperty
  {
   get;
   set;
  }

  public int SamplePublicField;

  SampleClass()
  {
   SampleReadonlyField = 1;
  }

  delegate void SampleDelegate();
  event SampleDelegate SampleEvent;

  public void SampleMethod()
  {
  }
 }
```

1.2 Camel casing

Описываются имена:

• локальных переменных;

• аргументов методов;
• защищенных (protected) полей.

```
protected int sampleProtectedField;

int SampleMethod(int sampleArgument)
{
  int sampleLocalVariable;
```

1.3 Суффиксы и префиксы

Применяются следующие суффиксы и префиксы:
• имена пользовательских классов исключений всегда заканчиваются суффиксом “Exception”;
• имена интерфейсов всегда начинаются с префикса «I»;
• имена пользовательских атрибутов всегда заканчиваются суффиксом «Attribute»;
• имена делегатов обработчиков событий всегда оканчиваются суффиксом EventHandler, имена классов-наследников от EventArgs всегда заканчиваются суффиксом EventArgs.

```
public class SampleException: System.Exception
{
  public SampleException()
  {
  }
}

interface ISample
{
  void SampleMethod();
}

[System.AttributeUsage(System.AttributeTargets.All, Inherited = false, AllowMultiple = true)]
sealed class SampleAttribute: System.Attribute
{
  public SampleAttribute()
  {
  }
}  public delegate void AnswerCreatedEventHandler(object sender, AnswerCreatedEventArgs
e);
  public class AnswerCreatedEventArgs: EventArgs
  {
    public int CreatedId;
    public int ParentId;
```

```
    public string CreatorName;
```

1.4 Аббревиатуры

При использовании аббревиатур в именах, капитализации подлежат аббревиатуры с двумя символами, в остальных аббревиатурах необходимо приводить к верхнему регистру только первый символ.

```
namespace Sample.IO
{
}

class HttpUtil
{
```

2. Рекомендации

2.1 Именование методов

Используйте конструкцию глагол-объект для именования методов

```
ShowUserInfo()
```

В частном случае, для методов, которые возвращают значение, используйте в паре глагол-объект для глагола «Get», а для объекта – описание возвращаемого значения.

```
GetUserId()
```

2.2 Переменные, поля и свойства

• При именовании переменных избегайте использования сокращенных вариантов вроде **I** и **t**, используйте **index** и **temp**. Не используйте венгерскую нотацию или используйте ее только для закрытых членов. Не сокращайте слова, используйте **number**, а не **num**.
• Рекомендуется для имен элементов управления указывать префиксы, описывающие тип элемента. Например: txtSample, lblSample, cmdSample или btnSample. Эта же рекомендация распространяется на локальные переменные сложных типов:
ThisIsLongTypeName tltnSample = new ThisIsLongTypeName();
• не используйте публичных или защищенных полей, вместо этого используйте свойства;

• используйте автоматические свойства;
• всегда указывайте модификатор доступа private, даже если разрешено его опускать;
• всегда инициализируйте переменные, даже когда существует автоматическая инициализация.

2.3 Дополнительные рекомендации

• используйте пустую строку между логическими секциями в исходном файле, классе, методе;
• используйте промежуточную переменную для передачи bool-значения результата функции в условное выражение **if**;

```
bool boolVariable = GetBoolValue();
if (boolVariable)
{
```

2.4 Объем кода

• избегайте файлов с более чем 500 строками кода;
• избегайте методов с более чем 200 строками кода;
• избегайте методов с более чем 5 аргументами, используйте структуры для передачи большого числа параметров;
• одна строка кода не должна иметь длину более 120 символов.

3. Спорные моменты для обсуждения

3.1 Закрытые поля

Первый вариант
Имена закрытых полей всегда начинаются с префикса «m_» остальная часть имени описывается с помощью Pascal Casing.

```
private int m_SamplePrivateField;
```

Второй вариант

Имена закрытых полей всегда начинаются с префикса «m» остальная часть имени описывается с помощью Pascal Casing.

private int mSamplePrivateField;

Третий вариант (пока самый вероятный на окончательный вариант) Имена закрытых полей всегда начинаются с префикса «_» остальная часть имени описывается с помощью Camel Casing.

private int _samplePrivateField;

3.2 Дополнительные требования

• всегда располагайте открывающие и закрывающие фигурные скобки на новой строке;
• всегда используйте фигурные скобки для выражения if, даже когда в выражение входит только одна строка кода.

Практическое задание. Используя предметные области (см. список ниже), выполните следующие действия:

1. Проверьте, насколько ранее созданный вами программный код соответствует правилам хорошего кодирования.
2. Внесите в программный код необходимые изменения.

Лабораторная работа №24-25. Разработка ПП: тестирование ПО

Теоретический материал.

1.Виды тестирования по целям

В зависимости от того, какие цели вы преследуете, тестируя ту или иную программу, тестирование бывает:

- **Функциональное.**
- **Нефункциональное.**

Функциональное тестирование направлено на проверку того, какие функции ПО реализованы, и того, насколько верно они реализованы.

Нефункциональное – проверяет корректность работы нефункциональных требований. Этот вид тестирования скорее проверяет, КАК программный продукт работает. Он включает в себя следующие виды:

- **Тестирование производительности** – проверяет как ПО работает под определенной нагрузкой.
- **Тестирование пользовательского интерфейса** – определяет удобство пользования разными параметрами интерфейса (кнопки, цвета, выравнивание и т. д.).
- **Тестирование удобства использования** – проверяет, удобен ли ПО в использовании.

- **Тестирование защищенности** – определяет, насколько безопасно использование программного продукта: защищено ли ПО от атак хакеров, несанкционированного доступа к данным и т. д.
- **Инсталляционное тестирование** – проверяет, не возникает ли проблем при установке, удалении, а также обновлении ПО.
- **Тестирование совместимости** – тестирование работы программного продукта в определенном окружении.
- **Тестирование надежности** – проверяет работу ПО при длительной средней ожидаемой нагрузке.
- **Тестирование локализации** – тестирование локализованной версии программного продукта (языковой и культурный аспекты).

2. По степени автоматизации

В зависимости от того, используют ли тестировщики дополнительные программные средства для тестирования приложений или программ, тестирование бывает:

- **Ручное** – без использования дополнительных программных средств, т. е. тестирование «вручную».
- **Автоматизированное** – с использованием программных средств

3. По позитивности сценария

По позитивности сценария тестирование бывает:

- **Позитивным** – проверка ПО на соответствие ожидаемому поведению. Это самый первый вид тестирования, который следует проводить, ведь основная задача тестирования – проверить, корректно ли работает программа.
- **Негативным** – проверяет, будет ли ПО работать в случае, когда поведение пользователя отличается от ожидаемого.

4. По доступу к коду программного продукта

В процессе тестирования инженер может работать с ПО, не обращаясь к его коду, а может определить правильность работы, взглянув на код. По доступу к коду программного продукта тестирование делится на:

- **Тестирование «белого ящика»** – тестирование программного продукта с доступом к коду.
- **Тестирование «черного ящика»** – тестирование без доступа к коду продукта.
- **Тестирование «серого ящика»** – тестирование, основанное на ограниченном знании внутренней структуры ПО. Часто говорят, что это смесь тестирования «белого ящика» и «черного ящика», но это в корне неверно. В данном случае тестировщик не работает с кодом программного продукта, но он знаком с внутренней структурой программы и взаимодействием между компонентами.

5. По уровню

По уровню тестирование бывает:

- **Модульное / юнит-тестирование** – проверка корректной работы отдельных единиц ПО. Этот вид тестирования могут выполнять сами разработчики.
- **Интеграционное тестирование** – проверка взаимодействия между несколькими единицами ПО.
- **Системное** – проверка работы всей системы на соответствие заявленным требованиям к программному продукту.

6. По исполнителю

Наверняка, вы слышали об альфа- и бета-тестировании. А поучаствовать в одном из них можно, даже не будучи тестировщиком. Итак, по исполнителю тестирование делится на:

- **Альфа-тестирование** – тестирование программного продукта на поздней стадии разработки. Проводится разработчиками или тестировщиками.
- **Бета-тестирование** – тестирование ПО перед выходом на рынок силами обычных людей – добровольцев, которым передается предварительная версия продукта (бета-версия). Их отзывы собираются, анализируются и учитываются при внесении правок в продукт.

7. По формальности

По формальности тестирование бывает:

- **Тестирование по тестам** – тестирование по предварительно написанным тест-кейсам.
- **Исследовательское тестирование** – одновременная разработка тестов и их исполнение.
- **Свободное тестирование** – тестирование без разработки тестов, без документации. Основывается на интуиции и опыте тестировщика.

8. По важности

По степени важности тестируемых функций тестирование делится на:

- **Дымовое тестирование** – проверка самой важной функциональности программного продукта.
- **Тестирование критического пути** – проверка функциональности, используемой типичными пользователями в повседневной деятельности.
- **Расширенное тестирование** – проверка всей заявленной функциональности.

Виды тестирования и подходы к классификации тестирования отличаются от автора к автору. Не существует единственного правильного варианта.

Практическое задание. Используя предметные области (см. список ниже), выполните следующие действия:

1. Проведите функциональное тестирование программного продукта.
2. Проведите тест удобства использования программного продукта.

3. Проведите тест на совместимость (кроссплатформенность).
4. Осуществите контрольный пример с использованием эталонных входных данных и предсказанных результатов.

   Результаты испытаний оформите в виде итогового протокола.

Лабораторная работа №26. Разработка ПП: оформление итоговой документации.

Теоретический материал.

Итоговый документ разработки ПО

Итоговый документ разработки ПО — основной документ по демонстрации соответствия Плану сертификации в части ПО. Этот документ должен содержать следующие разделы:

— Краткий обзор системы. Данный раздел содержит краткий обзор системы, включая описание ее функций и их распределение на программную и аппаратную реализацию, архитектуру, используемые процессоры, интерфейсы аппаратных средств/ПО, требования по обеспечению безопасности. Этот раздел также описывает все отличия от краткого обзора системы в Плане сертификации в части ПО.

— Краткий обзор ПО. Этот раздел кратко описывает функции ПО с акцентированием на обеспечении безопасности и используемой концепции разбиения и объясняет отличия от краткого обзора ПО в Плане сертификации в части ПО.

— Вопросы сертификации. Этот раздел вновь рассматривает вопросы сертификации, определенные в Плане сертификации в части ПО, и объясняет все существующие от указанного плана отличия.

— Характеристики ПО. В этом разделе указаны размер исполняемого объектного кода, ограничения по времени и памяти, ограничения ресурсов и способы измерения каждой характеристики.

— Жизненный цикл ПО. Этот раздел описывает фактически используемую модель жизненного цикла ПО и объясняет ее отличия от предложенной в Плане сертификации в части ПО.

— Документы жизненного цикла ПО. В этом разделе даны ссылки на документы жизненного цикла ПО, являющиеся выходными результатами процессов разработки ПО и интегральных процессов. Здесь описаны связь между представляемыми документами и другими документами, определяющими систему, а также способы передачи документов жизненного цикла ПО сертифицирующей организации. В этом разделе также рассмотрены любые отклонения в описании документов от Плана сертификации в части ПО.

— Идентификация ПО. Этот раздел идентифицирует конфигурацию ПО посредством указания регистрационного номера и версии.

— Хронология изменения. В случае необходимости этот раздел включает в себя резюме изменений ПО с указанием изменений, вызванных отказами, влияющими на безопасность, и идентификацией изменений, выполненных после предыдущей сертификации.
— Текущее состояние ПО. Этот раздел содержит резюме сообщений о дефектах, не устраненных ко времени сертификации, включая заявления о функциональных ограничениях.
— Утверждение о соответствии. Этот раздел включает в себя утверждение о соответствии требованиям настоящего стандарта и резюме методов, позволяющих показать выполнение критериев, определенных в планах ПО. Этот раздел также указывает дополнительные соглашения и отклонения от требований планов, стандартов разработки и настоящего стандарта.

Практическое задание. Используя предметные области (см. список ниже), выполните следующие действия:

1. Составьте список пунктов, которые должны присутствовать в итоговой документации конкретно для вашего программного продукта.
2. Заполните следующие пункты: «Краткий обзор ПО», «Характеристики ПО», «Жизненный цикл ПО», «Хронология изменений», «Текущее состояние ПО».

МДК.02.02 «Инструментальные средства разработки программного обеспечения»

Лабораторная работа №4. Исследование информационных потоков

Теоретический материал.

Важный этап в рационализации существующих систем управления – анализ потоков информации, который должен обеспечить выполнение целевых задач проектирования и уточнение особенностей существующей практики планирования.

Анализ существующих процессов управления может быть выполнен, прежде всего, на базе исследования информационной системы предприятия, которая характеризуется наличием существующей схемы документооборота, системы экономических показателей деятельности предприятия, структурным составом подразделений, участвующих в процессе управления, и интенсивностью потоков данных, циркулирующих между ними.

Обработанные материалы обследования позволяют провести анализ системы планирования и управления как в отдельных подразделениях управляющей системы, так и на предприятии в целом, а также создать предпосылки для построения стройной схемы обработки данных.

Деятельность любого подразделения, связанная с управлением, выражается в создании различных форм документов и показателей.

Анализ потоков информации позволяет выявить:

1) структуру и функции подразделений предприятия;

2) наименование различных подразделений и внешних организаций, с которыми взаимодействует данное подразделение;

3) перечень документов, поступающих в подразделения;

4) перечень документов, разрабатываемых в каждом подразделении;

5) перечень всех сообщений, поступающих в данное подразделение, с указанием, откуда поступило каждое из них;

6) перечень всех выходящих из данного подразделения документов (сообщений) с указанием их адреса;

7) перечень справочных данных и законодательных положений, используемых в работе подразделений;

8) четкое определение процессов формирования и маршрутов движения документов;

9) назначение форм документов;

10) количество разрабатываемых экземпляров форм документов;

11) периодичность составления документов;

12) основания и процессы принятия решений;

13) список показателей, содержащихся в каждом конкретном документе (сообщении), поступающем в подразделение;

14) список показателей, содержащихся в каждом конкретном документе, разрабатываемом в данном подразделении;

15) список показателей, содержащихся в каждом конкретном документе (сообщении), выходящем из данного подразделения;

16) повторение одинаковых или одноименных показателей в данном подразделении;

17) применяемость показателей;

18) оседаемость показателей в подразделении;

19) значность показателей.

Анализ потоков информации с точки зрения выше перечисленных задач позволяет получить материал для совершенствования существующей системы планирования и управления производством при разработке информационной системы предприятия (ИСП).

Для проектирования ИСП необходимо выяснить схему планирования и управления производством в существующих условиях; полный перечень показателей планирования, учета и управления производством на предприятии; типовые процедуры расчета показателей на предприятии; последовательность работ, связанных с планированием и управлением производством на промышленном предприятии.

Схема планирования и управления производством в существующих условиях, как представляется, выражается информационной моделью системы.

Очевидно, чтобы получить четкое представление о параметрах существующей системы планирования, в информационной модели необходимо устранить некоторые недостатки. Прежде всего, это относится к факторам, определяемым сложившейся системой документооборота. Другие факторы определяются существующей организационной структурой.

В процессе анализа информационных потоков выявляются все необходимые показатели, их роль в решении задач планирования и управления, а также необходимость их использования в условиях ИСП.

Выявленный перечень показателей подлежит анализу с точки зрения научного подхода к созданию экономического языка планирования и управления производством на промышленных предприятиях.

Как показывают предварительные исследования, все показатели планирования, учета и управления формируются на основе ограниченного числа типовых процедур, которые могут быть выявлены в процессе анализа. Определение этих процедур может обеспечить разработку стандартных программ для электронно-вычислительных машин.

Анализ информационных потоков – связующее звено между изучением существующей системы управления предприятием и ее совершенствованием.

Анализируя ИП, уточняют схему организации управления в исследуемой системе, что позволяет выявить эффективность существующей оргструктуры, узкие места в системе обработки данных и наметить пути к улучшению существующей организации управления предприятием. Можно сказать, что цель анализа ИП – выработка предложений по совершенствованию организации системы управления предприятием, которая проводится на основе исследования и обобщения всех материалов, полученных в ходе изучения системы и ее совершенствования.

Анализ ИП позволяет уточнить схему существующей структуры предприятия. Для этого необходимо рассмотреть все цепочки в системе обработки данных, начиная с получения исходных сведений; их постепенное преобразование и формирование конечных данных, которые направляются управляемой системе в качестве команд и внешним организациям в качестве отчетной и прочей информации. При этом определяется роль каждого подразделения в комплексе работ, выполняемых системой управления предприятием и

зафиксированных в схеме обработки данных; строится схема информационных связей между подразделениями предприятия. В схеме могут содержаться сведения о конкретных формах информационных связей и указываться их количественные и временные характеристики, а также определяться каналы связи, необходимые для передачи, и выявляться первичные (исходные) для предприятия данные. Первичными данными будем считать такие, которые поступают в систему управления из внешнего мира (планы, справки, патенты и т.д.), возникают в управляемой системе (например, данные измерений) или хранятся в памяти системы управления предприятием. Все эти сведения объединяются тем свойством, что они используются в работе системы управления предприятием, поступая в нее в готовом виде. При этом может быть построена логическая схема, характеризующая последовательные этапы обработки данных, исследована целесообразность имеющихся повторений некоторых сведений в системе обработки данных, для чего необходимо подробное рассмотрение существующих процедур обработки данных с использованием схемы информационных связей, а также определены первичные (исходные) показатели, необходимые для формирования каждого производного показателя.

Эти материалы используются:

• для организации системы памяти на предприятии;

• определения круга показателей (первичных и производных), необходимых каждому подразделению предприятия (каждому работнику) с целью выполнения их функций, связанных с обработкой данных;

• получения характеристик существующей системы обработки данных, таких как оседаемость показателей, степень их использования и др.

Получение необходимых материалов для анализа информационных потоков весьма трудоемкий процесс и требует для облегчения и ускорения этих работ использования вычислительных средств.

Система документооборота на предприятии является отображением его производственно-хозяйственной деятельности. По мере совершенствования производственного процесса происходило и изменение документооборота, которое выражалось в появлении новых (или ликвидации существующих) форм документов и изменении маршрутов их движения.

Таким образом, схема документооборота сложилась в результате длительного развития под влиянием объективных и субъективных причин.

Основные объективные причины, определяющие схему документооборота, заключаются в закономерности ведения производственного процесса и его совершенствования. Эти причины обусловливают основные принципы и общие черты систем управления, представляющих собой единый процесс обработки данных.

Такое положение характерно для систем управления любых типов предприятий с различными схемами обработки данных.

Конкретное отражение общего контура управления в настоящее время проявляется в системах документооборота, т.е. в маршрутах движения плановых и отчетных форм документов и использовании нормативно-справочных данных.

Субъективные причины накладывают определенный отпечаток на существующую схему документооборота, однако даже предварительное ознакомление может выявить в ней некоторую закономерность. Это, прежде всего, проявляется в наличии различной направленности движения плановых и отчетных форм документов, а следовательно, и процессов их образования и использования. При этом нормативные документы являются как бы "питающей" базой, т.е. используются для формирования обеих групп. Для проведения анализа системы обработки данных представляется целесообразным разделить все циркулирующие на промышленном предприятии документы на три основные группы: плановые, фактические (отчетные), нормативные.

Для построения комплексной схемы обработки данных анализ существующего документооборота должен проводиться согласно этой классификации. В данном случае анализ выполняется по отдельно построенным схемам плановых, фактических и нормативных документов, а также на примере общей совмещенной схемы документооборота станкостроительного предприятия.

Все это в итоге позволит выявить общие черты, присущие централизованной системе обработки данных, и выработать рекомендации по совершенствованию принципиальной схемы ИСП.

Анализ схемы движения плановых документов показывает, что отдельные задачи планирования на промышленном предприятии осуществляются самостоятельно подразделениями заводоуправления.

Хотя между подразделениями и существуют тесные взаимосвязи, которые усложняют (запутывают) процесс движения плановой документации при решении локальных плановых задач, все же основной поток данных поступает из подразделений в

производство, будучи регулятором его деятельности. Наличие связей и так называемых обратных связей между подразделениями объясняется следующими факторами: существующей методологией планирования; обособленностью подразделений в решении планово-экономических задач; децентрализованным использованием нормативов дли решения задач различных уровней планирования.

Это приводит к тому, что в процессе функционирования каждому подразделению приходится решать плановые задачи разных уровней. Такое положение в анализируемой схеме отображается различными цепочками циркулирующих документов и наличием обратных связей. Несмотря на это, все же можно выделить начало процесса планирования и проиллюстрировать его последовательность, осуществляемую различными подразделениями. Причем процесс формирования и движения плановых документов может быть представлен в двух аспектах: с наложением существующей организационной структуры и без нее.

При совершенствовании системы экономических показателей и рационализации схемы документооборота целесообразно уменьшить коэффициенты избыточности реквизитов. Для этого необходимо совершенствовать структуру форм документов как основных носителей информации:

• устранить из документооборота лишние формы, содержащие одну и ту же информацию;

• рационализировать схему документооборота на базе комплексности их обработки;

• сократить (по возможности) количество постоянных (нормативно-справочных) реквизитов в формах документов. Решение этих вопросов вызывает необходимость проведения классификации показателей и признаков, содержащихся в различных формах документов по группам.

По длительности изменения показатели могут быть разовые (переменные) и постоянные. Разовые показатели в неизменном виде существуют очень небольшой период времени (не более месяца) и меняются в документах одинаковой формы. Постоянные показатели устойчивы, и их числовое значение остается неизменным длительное время (цены, тарифы и т.д.), т.е. это всевозможные нормативно-расценочные и справочные показатели.

По способу образования все показатели можно подразделить на первичные и производные.

Показатели образуются тремя способами: путем учета и измерения; путем принятия заранее обусловленных норм, нормативов, расценок и тарифов; путем формирования новых показателей из уже известных.

Показатели, образованные двумя первыми способами, назовем первичными, показатели, формируемые из других, – производными.

По участию в обработке показатели можно подразделить на обрабатываемые и необрабатываемые. Обрабатываемые показатели, в свою очередь, делят на рабочие и вспомогательные.

По месту возникновения показатели можно подразделить на внутренние и внешние. Большой интерес представляет классификация показателей по функциям управления: плановые, учетноотчетные, конструкторско-технологические и т.д.

Признаки по участию в формировании показателя могут делиться на обязательные и необязательные. К обязательным относятся те, которые входят составной частью в показатель и образуют минимум части, образующей конкретный признак. Необязательные признаки не участвуют в формировании показателя, но они необходимы для последующих расчетов. По отношению к процессу обработки показателей признаки можно подразделить на группировочные, признаки-ограничители и справочные.

По отношению к технике регистрации и обработки информации можно выделить специальные, постоянные и переменные признаки.

Для проектирования форм документов и выбора носителей информации большое значение имеет деление признаков на постоянные и переменные.

Дальнейший анализ существующей системы сводится к выявлению перечня сводных показателей, на основе которых принимаются решения при выполнении тех или иных функций управления. Для каждого сводного показателя определяются процедуры их формирования и перечень первичных показателей, которые при этом используются.

После осуществления анализа информационных потоков наступает этап анализа системы обработки информации.

Пример картины информационных потоков компании:

Практическое задание.

1. Найдите ключевые подразделения организации, исходя из выбранной предметной области.
2. Составьте примерный список документов, которые имеют хождение в этой организации.
3. Составьте схему подразделений организации и связей между ними.

   Используя вышеприведённый образец, постройте схему информационных потоков вашей организации, с указанием документов между подразделениями.

Лабораторная работа №5. Построение модели IDEF1

Цель работы

Целью работы является освоение технологии построения информационной модели логического и физического уровней в нотации IDEF1X с использованием пакета Microsoft Visio.

Задачи

Основными задачами работы являются:

– приобретение студентами навыков построения информационной модели логического уровня;
– нормализации полученной модели;
– построения информационной модели физического уровня.

Теоретическая часть

1.1. Понятие информационной модели. Уровни информационной модели

Методология IDEF1X – язык для семантического моделирования данных, основанный на концепции
«сущность-связь».
Различают два уровня информационной модели: **логический** и **физический**.
Логическая модель позволяет понять суть проектируемой системы, отражая логические взаимосвязи между сущностями.
Различают 3 подуровня логического уровня модели данных, отличающиеся по глубине представления информации о данных:

– диаграмма сущность-связь (Entity-Relationship Diagram (ERD);
– модель данных, основанная на ключах (Key Based Model (KB);
– полная атрибутивная модель (Fully Attributed Model (FA).

Физическая модель отражает физические свойства проектируемой базы данных (типы данных, размер полей, индексы). Параметры физической информационной модели зависят от выбранной системы управления базами данных (СУБД).

1.2. Основные элементы информационной модели логического уровня

3.2.1. Сущности и атрибуты

Сущность – это множество **реальных** или **абстрактных объектов** (людей, предметов, документов и т.п.), **обладающих общими атрибутами или характеристиками**. Любой объект системы может быть представлен только одной сущностью, которая должна быть уникально идентифицирована. *Именование сущности* осуществляется с помощью *существительного в единственном числе*. При этом имя сущности должно отражать **тип** или **класс** объекта, а не его **конкретный экземпляр** (например, **Студент**, а не **Петров**) (рис. 1).

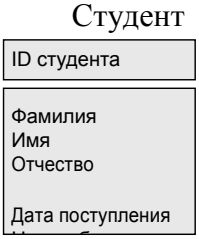


Рисунок 1 – Графическое представление сущности «Студент» в MS Visio

Любая сущность характеризуется набором атрибутов (**свойств**).
Атрибут сущности – характеристика сущности, то есть свойство реального объекта. Например, на рис. 1 атрибутами сущности **«Студент»** являются **«ID студента», «Фамилия», «Имя», «Отчество», «Дата поступления»** и **«Номер билета»**.

В свою очередь, *атрибуты сущности* делятся на 2 вида: *собственные* и *наследуемые. Собственные* атрибуты являются уникальными в рамках модели. *Наследуемые* атрибуты передаются от сущности-родителя при установке связи с другими сущностями.

Первичный ключ (Primary Key, PK). Каждая сущность должна обладать *атрибутом* или *комбинацией атрибутов*, чьи значения *однозначно определяют* каждый *экземпляр сущности*. Эти атрибуты образуют *первичный ключ* сущности.

Внешний ключ (Foreign Key, FK). Если между двумя сущностями *имеется специфическое отношение* связи или *категоризации*, то *атрибуты*, входящие в *первичный ключ родительской* или *общей сущности*, *наследуются* в качестве *атрибутов сущностью-потомком* или *категориальной сущностью* соответственно. Эти атрибуты и называются внешними ключами. Наследуемый атрибут может использоваться в сущности в качестве части или целого первичного ключа, альтернативного ключа или не ключевого атрибута.

3.2.2. Отношения в IDEF1X-модели

При построении информационной модели различают следующие типы отношений между сущностями:

идентифицирующее, не идентифицирующее, не специфическое (многие-ко-многим) и отношения категоризации.

Мощность отношения служит для обозначения отношения числа экземпляров родительской сущности к числу экземпляров дочерней.

1.3. Нормализация данных

Нормализация – это процесс проверки и реорганизации сущностей и атрибутов с целью удовлетворения требований к реляционной модели данных. Процесс нормализации сводится к последовательному приведению структур данных к нормальным формам – формализованным требованиям к организации данных.

Первая нормальная форма (1НФ). Сущность находится в первой нормальной форме тогда и только тогда, когда все атрибуты содержат атомарные значения. Среди атрибутов не должно встречаться повторяющихся групп, т.е. несколько значений для каждого экземпляра.

Вторая нормальная форма (2НФ). Сущность находится во второй нормальной форме, если она находится в первой нормальной форме, и каждый не ключевой атрибут полностью зависит от первичного ключа (не может быть зависимости от части ключа).

Третья нормальная форма (3 НФ). Сущность находится в третьей нормальной форме, если она находится во второй нормальной форме и никакой не ключевой атрибут не зависит от другого не ключевого атрибута (не должно быть зависимости между не ключевыми атрибутами).

Построение логической информационной модели уровня «сущность-связь»

1.4. Составление пула – списка потенциальных сущностей

Информационная модель может быть построена на основе функциональной модели или без нее. Использование функциональной модели в качестве основы для информационного моделирования позволяет создать структуру базы данных, полностью соответствующей функциям предприятия. Названия всех интерфейсных дуг функциональной модели (выполненной в нотации IDEF0) заносятся в *пул* – список потенциальных сущностей. Только в данном случае информационная модель будет адекватна выполняемым функциям. Функциональная модель для рассматриваемого примера представлена в *приложении А*.

Список потенциальных сущностей (при использовании программного продукта MS Office Visio для функционального моделирования) должен быть составлен вручную. В случае использования CASE-средства *AllFusion Process Modeler* отчет по интерфейсным дугам генерируется автоматически. Список потенциальных сущностей для рассматриваемого примера будет представлен таблицей вида (рис. 2).

Arrow Name
Варианты заданий
График
Графическая часть
Задание
Замечания, дополнения
Курсовая работа
Литература
Методические указания
Оценка за курсовую работу
Положение о курсовом проектировании
Пояснительная записка
Преподаватель
Расчеты
Список литературы
Студент

Рисунок 2 – Пул – список потенциальных сущностей

Теперь из этого списка необходимо выделить сущности, остальные интерфейсные дуги будут преобразованы в атрибуты сущностей.

В качестве сущностей выделим следующие:

1) задание;
2) пояснительная записка;
3) курсовая работа;
4) положение о курсовом проектировании;
5) студент;
6) преподаватель;
7) график;
8) методические указания.

1.5. Создание логической модели «сущность-связь»

1. Запустите MS Visio.
2. На закладке выбора шаблона выберите категорию *Программное обеспечение и базы данных* и в ней элемент *Схема модели базы данных*. Нажмите кнопку *Создать* в правой части экрана.
3. Установите необходимые параметры страницы (масштаб, ориентация страницы).
4. MS Visio поддерживает различные нотации моделей баз данных. Для того чтобы задать нотацию IDEF1X, необходимо выбрать пункты меню База данных → Параметры → Документ. В открывшемся окне на вкладке Общие установить переключатель в меню Набор символов на IDEF1X. Меню Имена, видимые на схеме, позволяет указать, какие имена атрибутов сущности будут отображены на диаграмме (концептуальные, физические или оба варианта одновременно). В данном случае для логического представления информационной модели необходимо выбрать пункт *Концептуальные имена* (рис. 3).

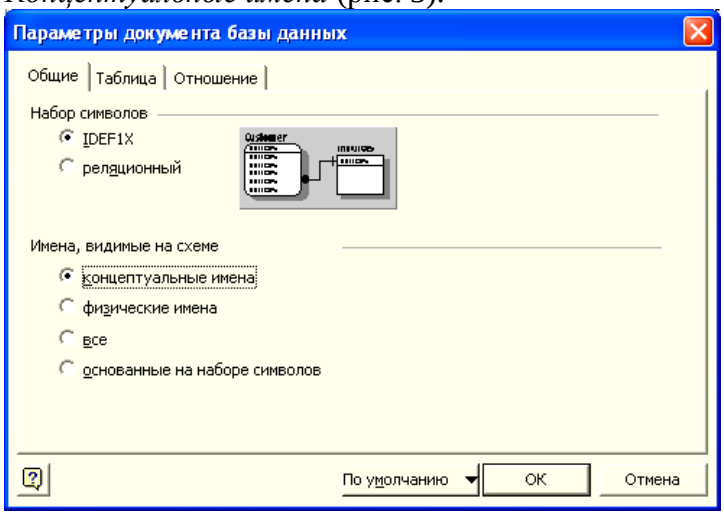


Рисунок 3 – Настройка параметров модели

В закладке *Отношение* окна *Параметры документа базы данных* в меню *Показывать* отметить галочкой пункт *Мощность*, в меню *Отображение вида* выбрать пункт

Показывать вербальную фразу, снять галочку в пункте *Обратный текст* (рис. 4). Данные настройки позволят отобразить имя и мощность связи в модели.

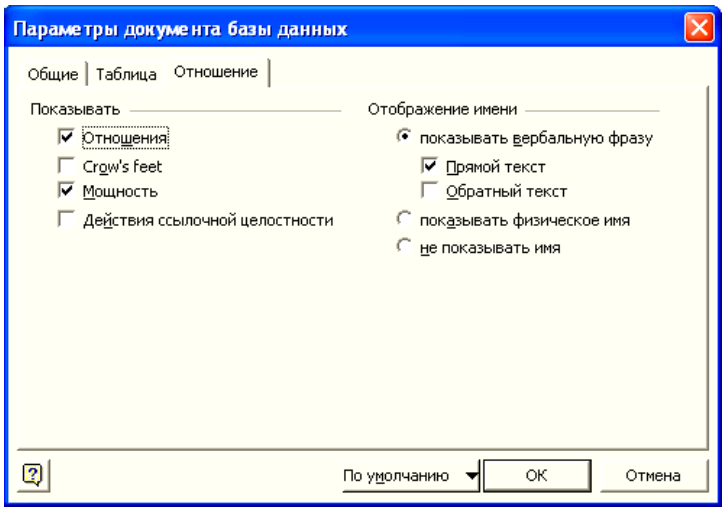


Рисунок 4– Настройка вида отношений информационной модели

5. Для того чтобы создать сущность, необходимо перетащить элемент на рабочее поле. Переход в режим редактирования сущности осуществляется двойным щелчком по сущности или по нажатию правой кнопки мыши и выбора пункта меню *Свойства базы данных*.

Чтобы задать имя сущности, в окне *Свойства базы данных* нужно выбрать категорию *Определение*, снять галочку в пункте *Синхронизация имен при вводе* (в противном случае, физическое и логическое имя сущности будут совпадать, что по практическим соображениям не всегда удобно) и задать концептуальное имя сущности. Руководствуясь данным алгоритмом, создадим 8 сущностей, определенных в пункте 5.1 (см. рис. 5).

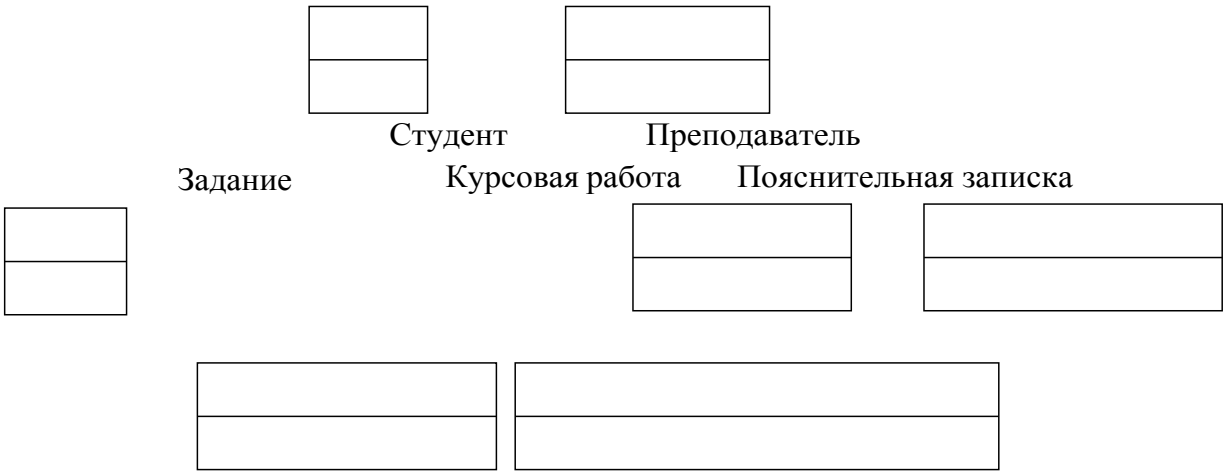


Рисунок 5– Сущности информационной модели логического уровня

6. Далее необходимо установить связи между сущностями.

Сначала составим описание предметной области на естественном языке. Любой студент должен выполнить одну или несколько курсовых работ. Каждая курсовая работа должна выполняться одним студентом (в идеале).

Каждая курсовая работа выполняется в соответствии с методическими указаниями и положением о курсовом проектировании.

Курсовая работа сдается по графику.

Курсовая работа оформляется в виде пояснительной записки.

Преподаватель проводит консультации, проверяет и ставит оценку за курсовую работу.

Таким образом, сформулируем имена связей:

СТУДЕНТ выполняет ***КУРСОВУЮ РАБОТУ***.
ПРЕПОДАВАТЕЛЬ проверяет ***КУРСОВУЮ РАБОТУ***.
КУРСОВАЯ РАБОТА выполняется в соответствии с ***ЗАДАНИЕМ***.
КУРСОВАЯ РАБОТА оформляется в виде ***ПОЯСНИТЕЛЬНОЙ ЗАПИСКИ***.
МЕТОДИЧЕСКИЕ УКАЗАНИЯ определяют требования к ***КУРСОВОЙ РАБОТЕ***.
КУРСОВАЯ РАБОТА организуется согласно ***ПОЛОЖЕНИЮ ПО КУРСОВОМУ ПРОЕКТИРОВАНИЮ***. ***КУРСОВАЯ РАБОТА*** сдается по ***ГРАФИКУ***.

Во всех случаях сущность *Курсовая работа* является дочерней, за исключением связи с сущностью *Пояснительная записка*. Определим типы связей и построим модель (см. рис. 6). В дальнейшем можно будет подкорректировать связи между сущностями.

Чтобы установить **связи** между сущностями, необходимо перетащить на рабочую область элемент  , поднести один конец стрелки к родительской сущности, другой – к дочерней.

Примечание. При правильном связывании каждая сущность будет подсвечена красным цветом.

В MS Office Visio по умолчанию используется *не идентифицирующее* отношение. Чтобы изменить **тип связи**, необходимо двойным щелчком по связи открыть окно *Свойства базы данных* и в категории в категории *Прочее* указать тип отношения (идентифицирующее, не идентифицирующее). В этой же категории указывается мощность связи (см. рис. 6).

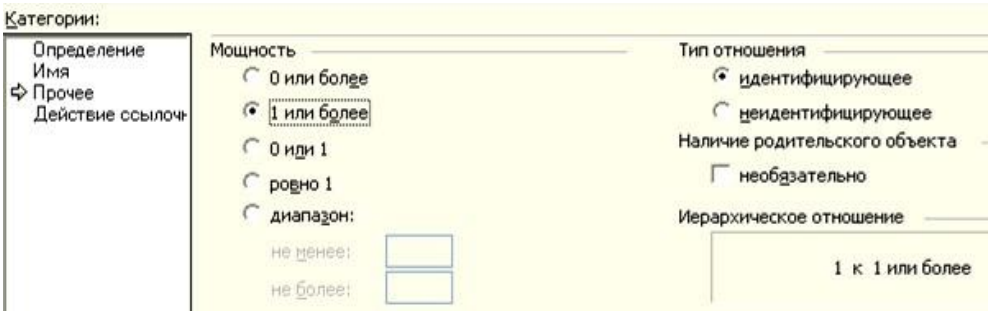


Рисунок 6 – Определение типа связи и мощности

Примечание. Кроме того, при не идентифицирующем отношении нужно указать, является ли наличие родительской сущности обязательным (т.е. может ли существовать экземпляр дочерней сущности, если не существует экземпляра родительской). Если наличие родительского объекта является необязательным, графически это отобразится в виде не закрашенного ромба со стороны родительской сущности.

Следующий шаг – в категории *Имя* в поле *Вербальная фраза* нужно указать имя отношения (рис. 7). Также можно указать имя связи в поле *Обратная фраза* для спецификации отношения потомок-родитель (в нашем случае обратная фраза отображаться не будет).

Примечание. Все изменения при закрытии окна свойств сохраняются автоматически.

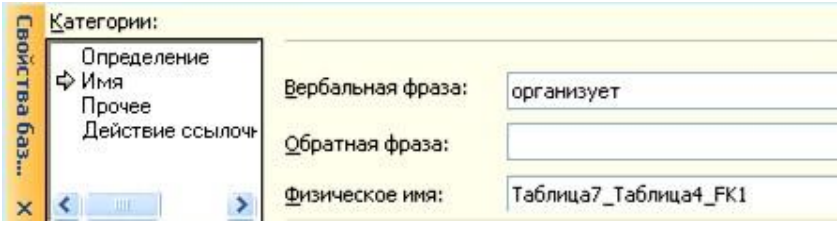


Рисунок 7 – Определение имени отношения

После определения имен, типов связей и задания мощностей получим информационную модель, представленную на рис. 8.

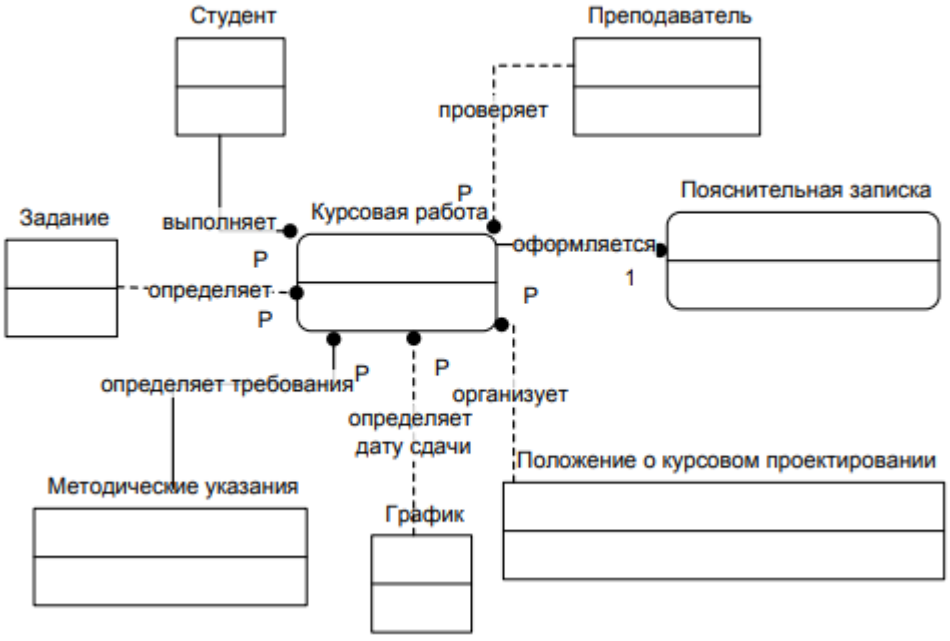


Рисунок 8 – Информационная модель уровня «сущность-связь»

Разработка логической модели данных, основанной на ключах

1. Необходимо определить ключевые атрибуты для каждой сущности, обращая внимание на то, что дочерние сущности наследуют ключевые атрибуты от родительских

(см. рис. 10).
Для этого двойным щелчком мыши по сущности откроем окно редактирования ее свойств, перейдем в категорию *Столбцы*, по нажатию кнопки *Добавить* введем имя поля (например, для сущности *Задание* ключевым атрибутом будет являться *Вариант задания*). Чтобы сделать атрибут ключевым, необходимо отметить галочкой пункт *PK* (рис. 9). Данное поле становится обязательным автоматически.

Рисунок 9 – Определение ключевого атрибута

Аналогичным образом зададим ключевые атрибуты для всех сущностей информационной модели. Результат представлен на рис. 10.
Как видно из рисунка 10 по сравнению с информационной моделью уровня «сущность-связь», был изменен тип связи между сущностями *Методические указания* и *Курсовая работа*, поскольку ключевые атрибуты сущности *Методические указания* для сущности *Курсовая работа* будут являться избыточными (зная номер зачетной книжки, можно узнать специальность и курс, на котором учится студент).

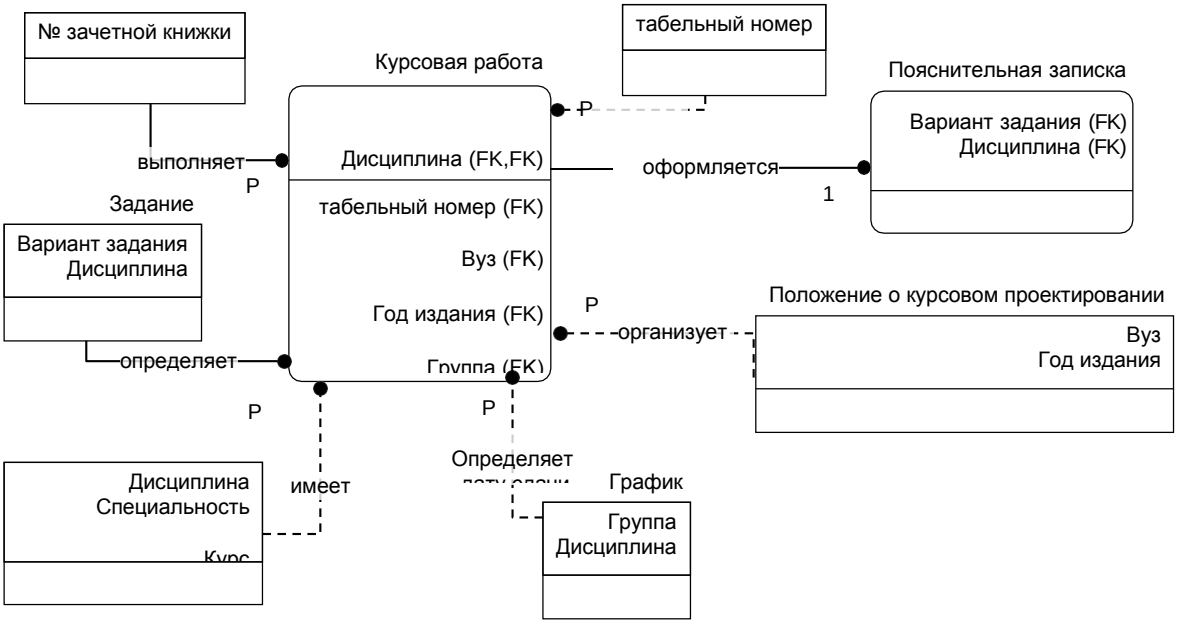


Рисунок 10 – Информационная модель с ключевыми атрибутами

2. Кроме того, отметим, что три сущности (Задание, График, Методические указания) содержат одинаковые атрибуты Дисциплина. Это является некорректным. Чтобы устранить данную ошибку, выделим одноименную сущность и свяжем ее идентифицирующими связями с вышеуказанными сущностями (рис. 11).

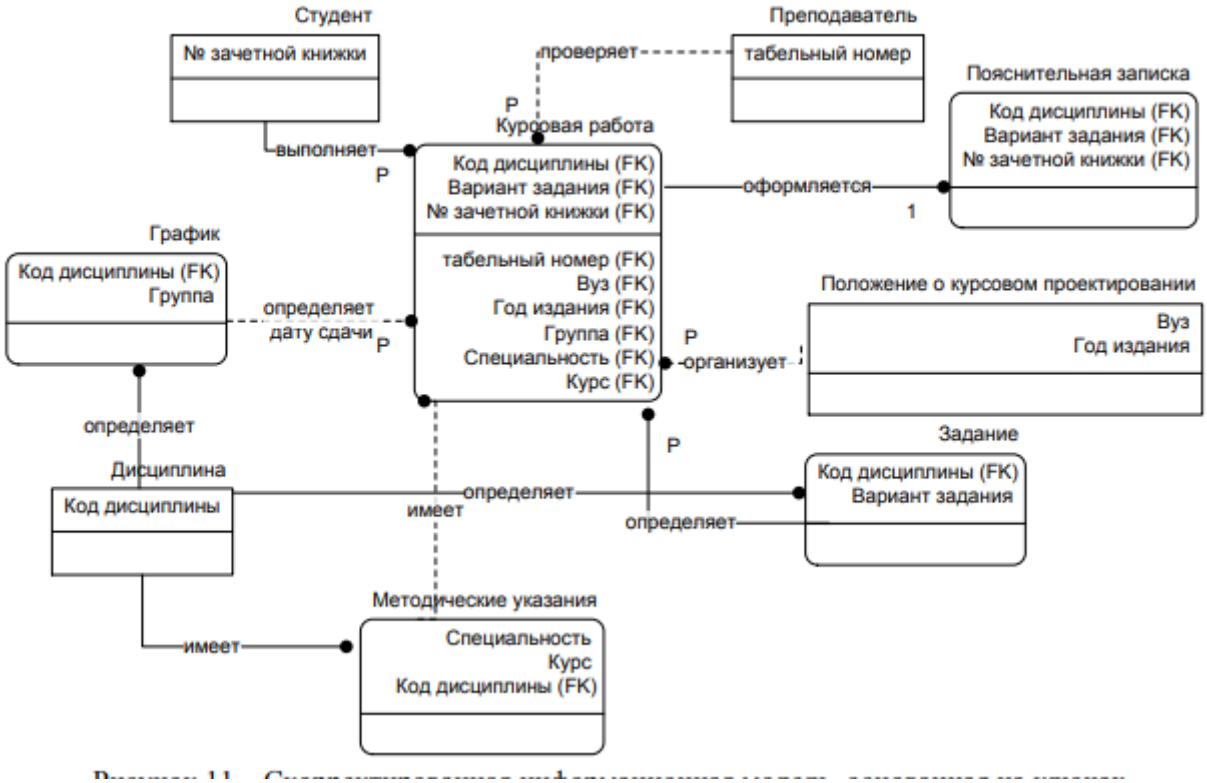


Рисунок 11 – Скорректированная информационная модель, основанная на ключах

Создание полной атрибутивной модели

Для того чтобы получить полную атрибутивную модель, необходимо дополнить сущности не ключевыми атрибутами. Дополненная модель представлена на рисунке 11.

Примечание. Если атрибут не является обязательным, нужно убедиться, что в окне *Свойства базы данных* в категории *Столбцы* в пункте *Обязательное* не стоит галочка. Не обязательные к заполнению атрибуты справа от имени имеют пометку *(О)*.

Создание физической модели

1. Необходимо переключиться на физический уровень представления информационной модели. Для этого нужно выбрать пункты меню *База данных* → *Параметры* → *Документ*. В открывшемся окне на вкладке *Общие* установить переключатель в меню *Имена, видимые на схеме*. В данном случае для физического представления информационной модели необходимо выбрать пункт *Физические имена* (рис. 14).
2. В закладке *Таблица* окна *Параметры документа базы данных* в меню *Отображать* выбрать пункт *Вертикальные линии,* в меню *Типы данных – Показывать физические* и в меню *Порядок – Физический порядок* (рис. 15).

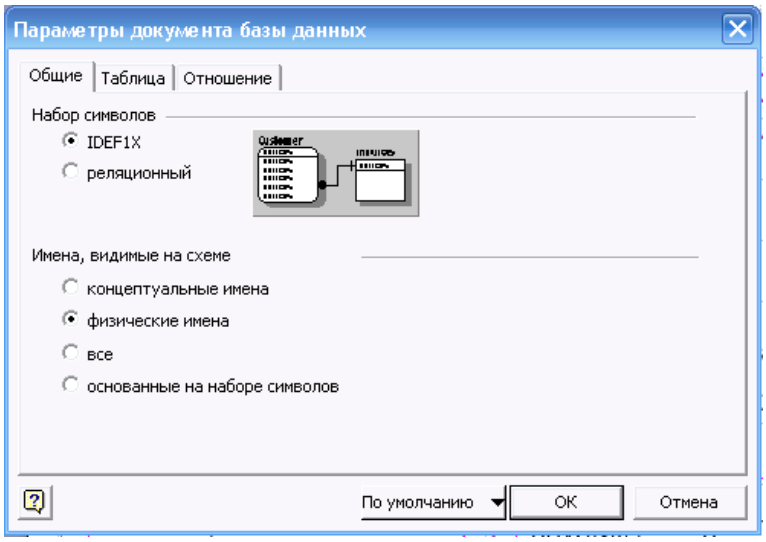


Рисунок 14 – Настройка параметров модели

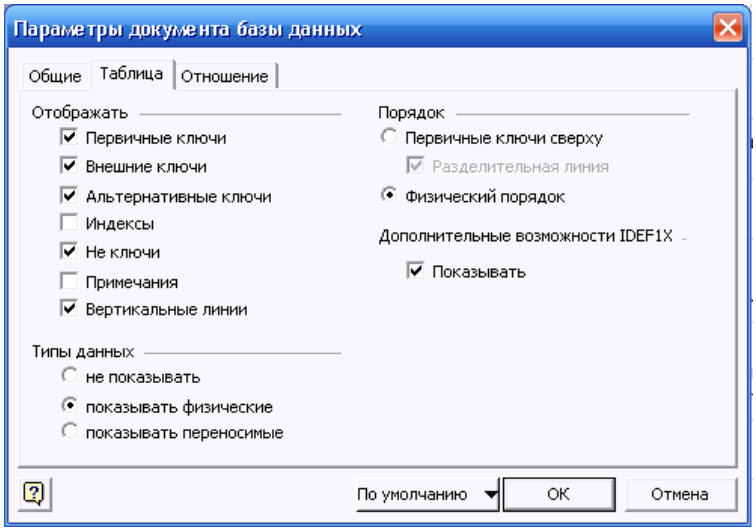


Рисунок 15 – Настройка параметров отображения сущности

3. В закладке Отношение окна Параметры документа базы данных в меню Отображение вида выбрать пункт Показывать физическое имя (рис. 16).

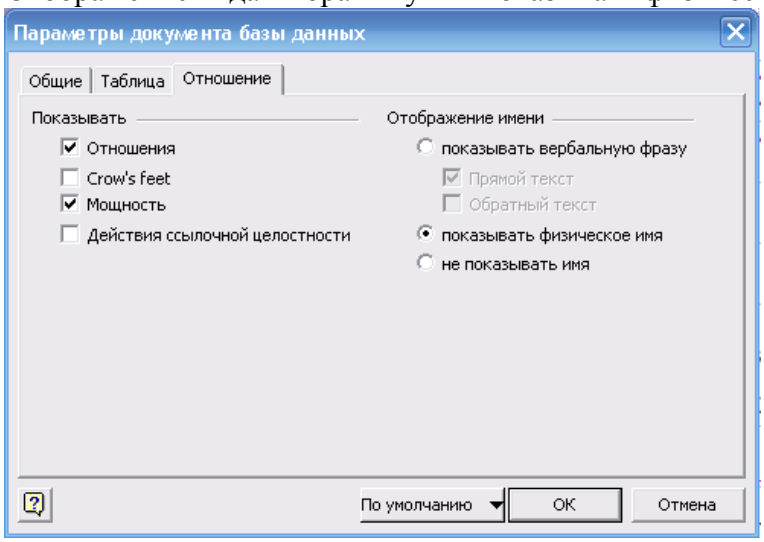


Рисунок 16 – Настройка вида отношений информационной модели

По окончании настройки документа информационная модель будет выглядеть, как представленная на рис. 17.

4. Для каждого атрибута (поля) необходимо определить тип данных.

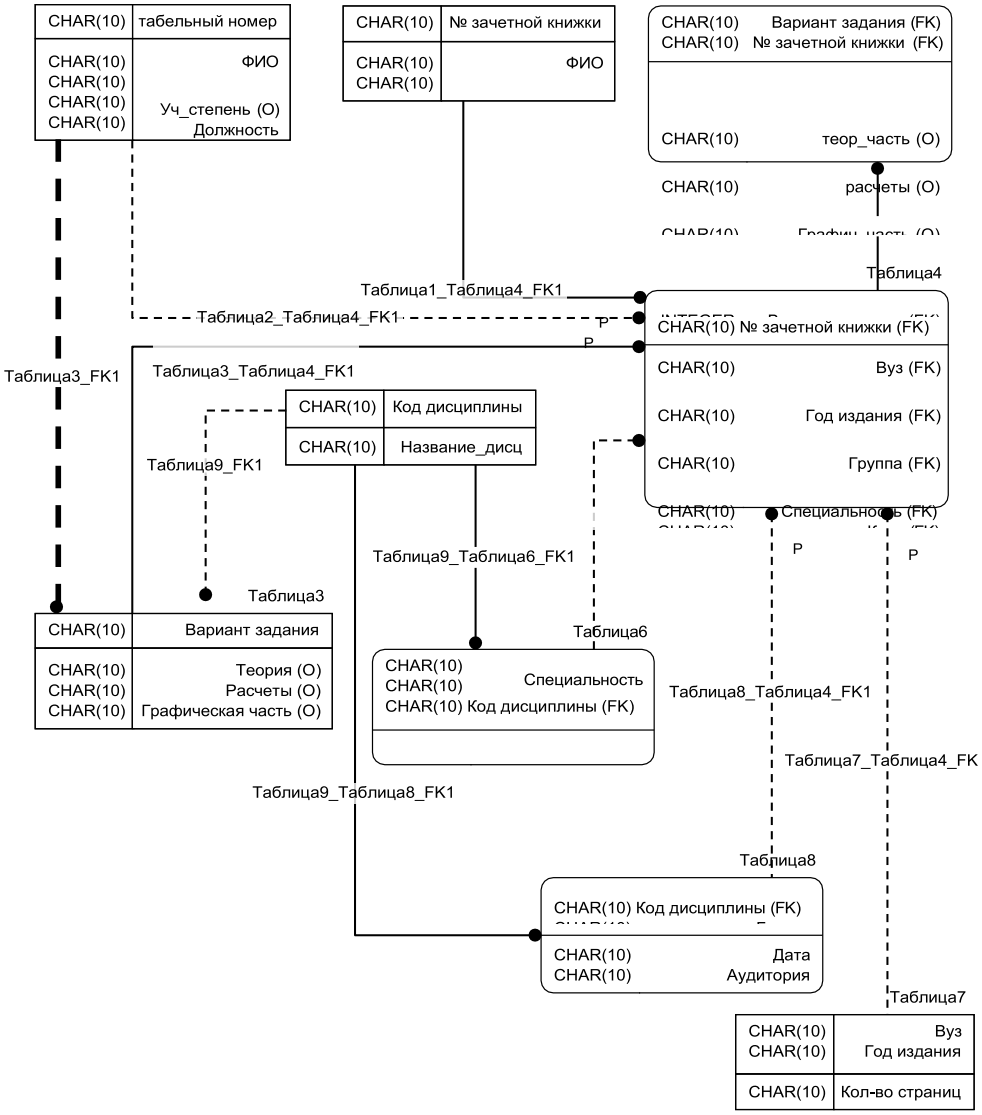


Рисунок 17 – Вид физической модели

Типы данных можно представить в виде правил, ограничивающих вид сведений, которые могут быть введены в каждый столбец таблицы базы данных. Например, чтобы в поле, которое предназначено только для дат, нельзя было ввести имя, этому полю назначается тип данных «Дата».

Примечание *(Выбор между переносимыми и физическими типами данных).*

Переносимые типы данных — это обобщенные типы данных, соответствующие в разных системах баз данных простым, совместимым между собой физическим типам.

Физические типы данных — это типы данных, поддерживаемые целевой базой данных.

Щелкните сущность, содержащую атрибуты, для которых требуется установить типы данных. В окне *Свойства базы данных* в списке *Категории* выберите вариант *Столбцы*. Под списком столбцов установите переключатель в положение *Физический тип данных*. В группе *Тип данных* для каждого атрибута выберите необходимый вариант из множества альтернатив (рис.

18). Описание типов данных приведено в *Приложении Б.*

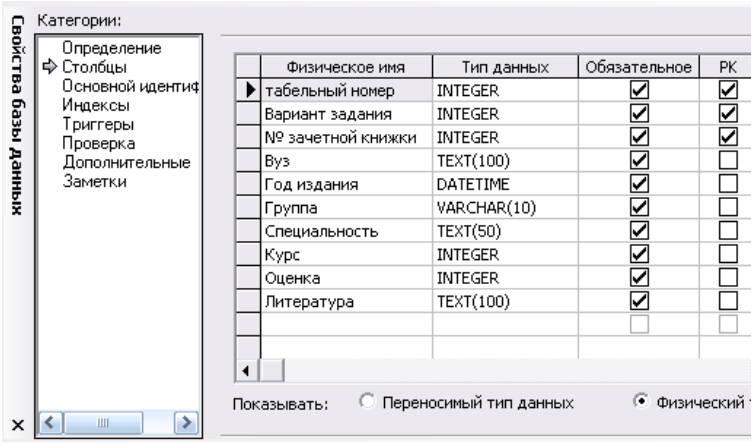


Рисунок 18– Определение типа данных атрибутов сущности

После того, как будут выполнены все действия, физическая модель будет выглядеть, как показано на рис. 19.

Таким образом, проделав все вышеперечисленные действия, получим информационную модель физического уровня, на основе которой может быть сгенерирована схема БД.

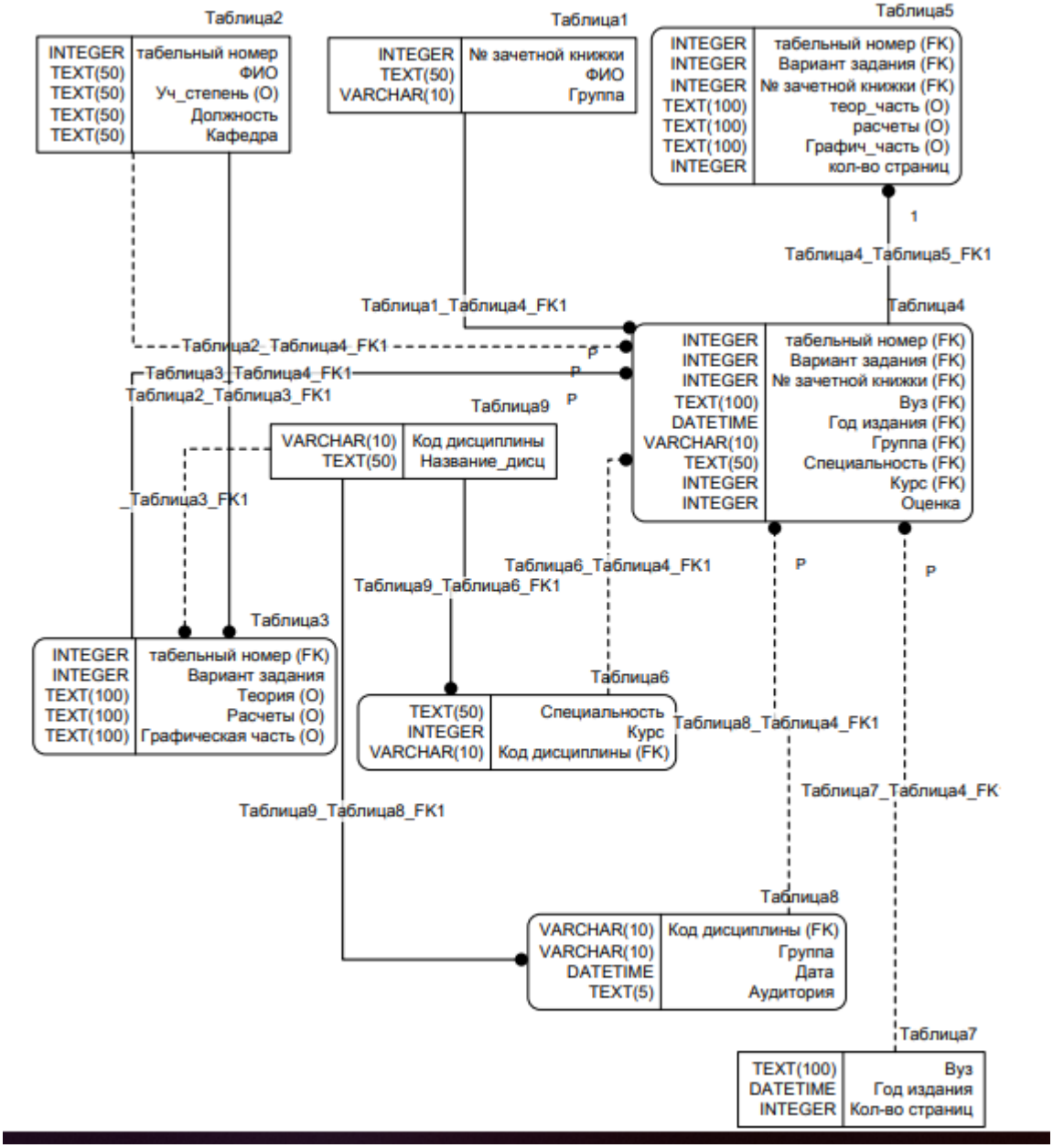


Рисунок 19 – Физическая модель базы данных

Практическое задание. Используя выбранную предметную область, выполните следующие действия:

5. Создать информационную модель логического уровня (выполнить упражнения 1-3). Минимальное количество сущностей – 4.
6. Провести нормализацию полученной модели.

   На основе нормализованной логической модели построить информационную модель физического уровня.

Лабораторная работа №7. Изучение процессов в системе

Цель работы

Целью работы является освоение технологии изучения процессов в конкретной предметной области.

Теоретические сведения

Исследования процессов, протекающих в технологических установках, установление закономерностей их протекания, нахождение зависимостей, необходимых для их анализа и расчета, можно проводить разными методами: *теоретическим, экспериментальным, подобия.*

Теоретический метод основан на составлении и решении системы дифференциальных уравнений, описывающих процесс. Дифференциальные уравнения описывают целый класс однородных по своей сущности явлений (процессов), поэтому для выделения конкретного явления необходимо ввести определенные ограничения, которые однозначно будут характеризовать данное явление. Эти дополнительные условия называются условиями однозначности. Условия однозначности включают в себя: геометрическую форму и размеры системы, т.е. аппарата, канала и т.д.; физические свойства веществ, участвующих в процессе; начальные условия (начальную температуру, начальную скорость и т.д.); граничные условия, например скорость жидкости у стенок канала, равную нулю.

Однако многие процессы химической технологии так сложны, что удается лишь составить систему дифференциальных уравнений и установить условия однозначности. Решить эти уравнения известными в математике методами не представляется возможным.

Экспериментальный метод позволяет на основе опытных данных получить эмпирические уравнения, описывающие данный процесс. Сложности экспериментального метода заключаются в необходимости проведения большого количества опытов на реальных технологических установках. Это связано с большими затратами средств и времени. Вместе с тем результаты проведенных экспериментов будут справедливы только для тех условий, для которых они получены, и не могут быть с достаточной надежностью перенесены на процессы, аналогичные изученным, но протекающие в других аппаратах.

Метод теории подобия позволяет с достаточной для практики точностью изучать сложные процессы на более простых моделях, обобщать результаты опытов и получать

закономерности, справедливые не только для данного процесса, но и для всей группы подобных процессов. При моделировании процессов можно вместо дорогостоящих трудоемких опытов на промышленных установках проводить исследования на моделях значительно меньших размеров, а вместо зачастую опасных и вредных веществ использовать безопасные модельные вещества, опыты проводить в условиях, отличных от производственных. Кроме того, материальную модель можно заменить физической схемой (моделью), отражающей существенные особенности данного процесса. Поэтому рассмотрим более подробно теории подобия.

Метод обобщенных переменных составляет основу теории подобия. Одним из основных принципов теории подобия является выделение из класса явлений (процессов), описываемых общим законом (процессы движения жидкостей, диффузии, теплопроводности и т.п.), *группы подобных явлений.*

Подобными называются такие явления, для которых отношения сходственных и характеризующих их величин постоянны.

Различают следующие виды подобия: геометрическое; временное; физических величин; начальных и граничных условий.

Технологический процесс (*сокращенно ТП*) — это упорядоченная последовательность взаимосвязанных действий, выполняющихся с момента возникновения исходных данных до получения требуемого результата.

Практически любой технологический процесс можно рассматривать как часть более сложного процесса и совокупность менее сложных (в пределе — элементарных) технологических процессов.

Элементарным технологическим процессом или технологической операцией называется наименьшая часть технологического процесса, обладающая всеми его свойствами. То есть это такой ТП, дальнейшая декомпозиция которого приводит к потере признаков, характерных для метода, положенного в основу данной технологии. Как правило, каждая технологическая операция выполняется на одном рабочем месте не более, чем одним сотрудником. Примером технологических операций могут служить ввод данных с помощью сканера штрих-кодов, распечатка отчета, выполнение SQL-запроса к базе данных и т. д.

Технологические процессы состоят из «технологических (рабочих) операций», которые, в свою очередь, складываются из «технологических переходов».

Этапы ТП

Технологический процесс обработки данных можно разделить на четыре укрупненных этапа:

- «Начальный или первичный». Сбор исходных данных, их регистрация (прием первичных документов, проверка полноты и качества их заполнения и т. д.) По способам осуществления сбора и регистрации данных различают следующие виды ТП:

- механизированный — сбор и регистрация информации осуществляется непосредственно человеком с использованием простейших приборов (весы, счетчики, мерная тара, приборы учета времени и т. д.);
- автоматизированный — использование машиночитаемых документов, регистрирующих автоматов, систем сбора и регистрации, обеспечивающих совмещение операций формирования первичных документов и получения машинных носителей; автоматический — используется в основном при обработке данных в режиме реального времени (информация с датчиков, учитывающих ход производства — выпуск продукции, затраты сырья, простои оборудования — поступает непосредственно в ЭВМ).

- «Подготовительный». Прием, контроль, регистрация входной информации и перенос её на машинный носитель. Различают визуальный и программный контроль, позволяющий отслеживать информацию на полноту ввода, нарушение структуры исходных данных, ошибки кодирования. При обнаружении ошибки производится исправление вводимых данных, корректировка и их повторный ввод.
- «Основной». Непосредственно обработка информации. Предварительно могут быть выполнены служебные операции, например, сортировка данных.
- «Заключительный». Контроль, выпуск и передача результатной информации, её размножение и хранение.

Практическое задание. Используя выбранную предметную область, выполните следующие действия:

1. Выясните подробно, сколько участников выполняют различные действия в предметной области.
2. Выберите наиболее характерные для них действия и составьте список этих действий.
   Составьте схему технологии выполнения этих действий, соблюдая логику и последовательность. Условие: **необходимо составить не менее трёх технологических процессов,** которые представляют собой полную карту выполнения конкретных задач эктора (участника) информационной системы предметной области.

Лабораторная работа №8-9. Построение модели IDEF3

Цель работы

Целью работы является освоение технологии построения модели технологических процессов предметной области в нотации IDEF3 с использованием пакета Microsoft Visio или его аналога.

Задачи

Основными задачами практической работы являются:

приобретение студентами навыков построения модели технологических процессов.

Краткие теоретические сведения

IDEF3 является стандартом документирования технологических процессов, происходящих на предприятии, и предоставляет инструментарий для наглядного исследования и моделирования их сценариев**. Сценарием (Scenario)** мы называем описание последовательности изменений свойств объекта, в рамках рассматриваемого процесса (например, описание последовательности этапов обработки детали в цеху и изменение её свойств после прохождения каждого этапа). Исполнение каждого сценария сопровождается соответствующим документооборотом, который состоит из двух основных потоков: документов, определяющих структуру и последовательность процесса (технологических указаний, описаний стандартов и т.д.), и документов, отображающих ход его выполнения (результатов тестов и экспертиз, отчетов о браке, и т.д.). Для эффективного управления любым процессом, необходимо иметь детальное представление об его сценарии и структуре сопутствующего документооборота. Средства документирования и моделирования IDEF3 позволяют выполнять следующие задачи:

Документировать имеющиеся данные о технологии процесса, выявленные, скажем, в процессе опроса компетентных сотрудников, ответственных за организацию рассматриваемого процесса.
Определять и анализировать точки влияния потоков сопутствующего документооборота на сценарий технологических процессов.
Определять ситуации, в которых требуется принятие решения, влияющего на жизненный цикл процесса, например изменение конструктивных, технологических или эксплуатационных свойств конечного продукта.
Содействовать принятию оптимальных решений при реорганизации технологических процессов.
Разрабатывать имитационные модели технологических процессов, по принципу "КАК БУДЕТ, ЕСЛИ..."

Два типа диаграмм в IDEF3

Существуют два типа диаграмм в стандарте IDEF3, представляющие описание одного и того же сценария технологического процесса в разных ракурсах. Диаграммы, относящиеся к первому типу, называются **диаграммами Описания Последовательности Этапов Процесса (Process Flow Description Diagrams, PFDD)**, а ко второму - **диаграммами Состояния Объекта в и его Трансформаций Процессе (Object State Transition Network, OSTN)**. Предположим, требуется описать процесс окраски детали в производственном цеху на предприятии. С помощью диаграмм PFDD документируется последовательность и описание стадий обработки детали в рамках исследуемого технологического процесса.

Диаграммы OSTN используются для иллюстрации трансформаций детали, которые происходят на каждой стадии обработки.

На следующем примере, опишем, как графические средства IDEF3 позволяют документировать вышеуказанный производственный процесс окраски детали. В целом, этот процесс состоит непосредственно из самой окраски, производимой на специальном оборудовании и этапа контроля ее качества, который определяет, нужно ли деталь окрасить заново (в случае несоответствия стандартам и выявления брака) или отправить ее в дальнейшую обработку.

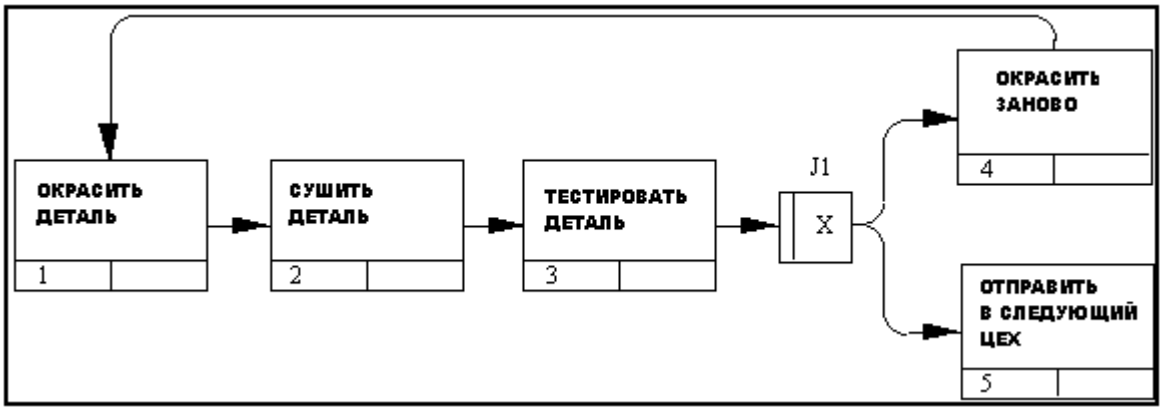


Рисунок 1. Пример PFDD диаграммы.

На рис.1 изображена диаграмма PFDD, являющаяся графическим отображение сценария обработки детали. Прямоугольники на диаграмме PFDD называются **функциональными элементами** или элементами поведения (Unit of Behavior, UOB) и обозначают событие, стадию процесса или принятие решения. Каждый UOB имеет свое имя, отображаемое в глагольном наклонении и уникальный номер. Стрелки или **линии** являются отображением перемещения детали между UOB-блоками в ходе процесса. Линии бывают следующих видов:

- Старшая (Precedence) - сплошная линия, связывающая UOB. Рисуется слева направо или сверху вниз.

- Отношения (Relational Link)- пунктирная линия, использующаяся для изображения связей между UOB

- Потоки объектов (Object Flow)- стрелка с двумя наконечниками используется для описания того факта, что объект (деталь) используется в двух или более единицах работы, например, когда объект порождается в одной работе и используется в другой.

Объект, обозначенный J1 - называется **перекрестком** (Junction). Перекрестки используются для отображения логики взаимодействия стрелок (потоков) при слиянии и разветвлении или для отображения множества событий, которые могут или должны быть

завершены перед началом следующей работы. Различают перекрестки для слияния (Fan-in Junction) и разветвления (Fan-out Junction) стрелок. Перекресток не может использоваться одновременно для слияния и для разветвления. При внесении перекрестка в диаграмму необходимо указать тип перекрестка. Классификация возможных типов перекрестков приведена в таблице.

Обозначение	Наименование	Смысл в случае слияния стрелок (Fan-in Junction)	Смысл в случае разветвления стрелок (Fan-out Junction)
	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
	Asynchronous OR	Один или несколько предшествующих процессов должны быть завершены	Один или несколько следующих процессов должны быть запущены
	Synchronous OR	Один или несколько предшествующих процессов завершаются одновременно	Один или несколько следующих процессов запускаются одновременно

X	XOR (Exclusive OR)	Только один предшествующий процесс завершен	Только один следующий процесс запускается

Все перекрестки в PFDD диаграмме нумеруются, каждый номер имеет префикс "J".

Практическое задание. Используя выбранную предметную область и результаты предыдущей работы, выполните следующие действия:

1. Выберите программное обеспечение, с помощью которого вы будете строить диаграмму IDEF3.
2. На основании созданных в предыдущей лабораторной работе схем технологических процессов, создайте три диаграммы IDEF3, соблюдая логику и последовательность.

Лабораторная работа №10. Построение UML-диаграмм классов.

Теоретический материал.

Как класс изображается на диаграмме UML?

Архитектор программного обеспечения в первую *очередь* обращает внимание на объекты *предметной области*. Программист же концентрируется на поведении этих объектов, пользуясь **классами**, к которым они принадлежат. Вот поэтому-то *диаграмма* классов и является одной из важнейших диаграмм *UML*. Она используется для документирования программных систем, и основным ее компонентом является **класс**. Что такое *класс*, мы уже говорили ранее, когда знакомились с видами диаграмм *UML*. В предыдущей лекции мы рассматривали назначение *диаграммы классов*, знакомились с примерами готовых диаграмм, но не вникали в тонкости обозначений, используемых на диаграмме. В тех примерах все казалось нам очень понятным и логичным. Тем не менее, некоторые нюансы все же следует рассмотреть, и как раз этим мы сейчас и займемся.

Класс на диаграмме изображается в виде прямоугольника, разделенного горизонтальными линиями на три части. В первой части указывается название класса. Как правило, *имя класса* состоит из одного, *максимум* двух слов. Вторая часть содержит перечень атрибутов класса, которые характеризуют тот или иной *объект* этого класса в модели *предметной области*. Третья часть содержит перечень операций, отражающих его поведение в модели *предметной области* (рис. 1). Все очень просто, не так ли?

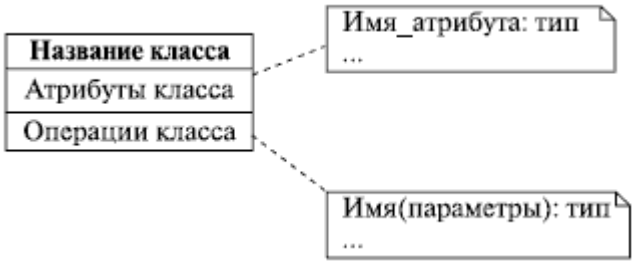


Рис. 1.

А что внутри?

Мы узнали, как *класс* изображается и выглядит "снаружи". А что же внутри объектов класса? Пользователю об этом знать необязательно, более того, абсолютно не нужно. Для человека, использующего его, *объект* выступает в роли черного ящика. Скрывая от пользователя внутреннее устройство объекта, мы обеспечиваем его надежную работу. Сейчас мы рассмотрим, как убрать из поля зрения пользователя то, что ему знать не нужно.

Читателя может слегка смутить *слово* "*пользователь*", которым мы злоупотребляли в предыдущем абзаце. Зачем вообще пользователю какие-то объекты и классы? Внесем *ясность*. Программист, использующий в своей программе созданные кем-то компоненты, как раз и выступает в роли такого пользователя. Зачем ему знать что внутри - он знает, какие атрибуты надо модифицировать и какие *операции* использовать, чтобы заставить *объект* работать именно так, как ему нужно! Более того, а многие ли из нас знают, как именно устроен и по каким принципам работает, например, *телевизор* - *объект* класса "Бытовой прибор"?

Сокрытие от пользователя внутреннего устройства объектов называется *инкапсуляцией*. Если говорить более "научным" языком, то *инкапсуляция* - это защита отдельных элементов объекта, не затрагивающих существенных характеристик его как целого. *Инкапсуляция* нужна не только для того, чтобы создать иллюзию простоты объекта для пользователя (по словам Г. Буча). Но вернемся к примеру с телевизором. Нам этот прибор кажется очень простым только потому, что при работе с ним мы используем простой и понятный *интерфейс* - пульт дистанционного управления. Мы знаем: для того чтобы увеличить громкость звука, надо нажать вот эту кнопку, а чтобы переключить канал - вот эту. Как *телевизор* устроен внутри, мы не знаем. Более того - в отсутствие пульта ДУ такое *знание* было бы неудобным для нас и весьма опасным для самого телевизора, вздумай мы увеличить громкость с помощью паяльника. Поэтому-то пульт ДУ и защищает от нас "внутренности" телевизора! Вот так *инкапсуляция* реализуется в реальном мире.

В программировании *инкапсуляция* обеспечивается немного по-другому - с помощью т. н. *модификаторов видимости*. С их помощью можно ограничить *доступ* к атрибутам и операциям объекта со стороны других объектов. Звучит это немного пугающе, но на самом деле все просто. Если *атрибут* или операция описаны с модификатором private, то *доступ* к ним можно получить только из *операции*, определенной в том же классе. Если же *атрибут* или операция описаны с модификатором видимости public, то к ним можно получить *доступ* из любой части программы. Модификатор protected разрешает *доступ* только из операций этого же класса и классов, создаваемых на его основе. В языках программирования могут встречаться *модификаторы видимости*, ограничивающие *доступ* на более высоком уровне, например, к классам или их группам, однако смысл инкапсуляции от этого не изменяется. В *UML* атрибуты и *операции* с модификаторами доступа обозначаются специальными символами слева от их имен:

Символ	**Значение**
+	public - открытый доступ
-	private - только из операций того же класса
#	protected - только из операций этого же класса и классов, создаваемых на его основе

Рассмотренный ранее пример с телевизором средствами *UML* (конечно же, это очень высокоуровневая *абстракция*) можно изобразить так (рис. 2):

Рис. 2.

Не правда ли, все понятно и предельно просто? Зачем, например, пользователю знать числовые значения частот каналов? Он знает, что достаточно запустить процедуру автоматического поиска каналов и *телевизор* все сделает за него. Вот вам и *инкапсуляция* - оказывается, она повсюду вокруг нас. Оглянитесь и подумайте, сколько вещей вокруг имеют скрытые свойства и выполняют скрытые *операции*. Испугались? Вот то-то же!

Как использовать объекты класса?

Итак, мы рассмотрели инкапсуляцию - одно из средств защиты объектов. Все вроде бы понятно, но как же именно работать с объектом?

Если уж говорить о защите объекта, то чтобы она действительно была эффективной, надо позаботиться о некоем стандартном и безопасном, не зависящим от языка программирования способе доступа к объекту. К тому же такой стандартный способ доступа должен быть простым и с точки зрения использования, и с точки зрения реализации. Вспомните пример с телевизором. Нажимая кнопки на пульте, мы ожидаем, что *телевизор* откликнется на это действие каким-то определенным образом - именно так, как мы ожидаем, а не иначе. То есть, с одной стороны, пульт ДУ является средством доступа к скрытым операциям, выполняемым телевизором, а с другой стороны - пульт обеспечивает нужное для нас поведение телевизора. В данном примере именно пульт является таким стандартным средством доступа к телевизору. Можно даже сказать, средством доступа, не зависящим от конкретной модели телевизора - вспомните об универсальных пультах и о том, как отключаете звук надоедливой рекламы на экране в вагоне поезда, используя КПК!

В том же примере с телевизором у нас впервые промелькнуло *слово интерфейс*. И не случайно промелькнуло: именно так называют тот самый стандартный способ доступа к объекту. Более строго, *интерфейс* - это логическая *группа* открытых (public) операций объекта. Один и тот же *объект* может иметь несколько интерфейсов. У телевизора, например, их два - пульт ДУ и кнопки на корпусе. А может и больше - вспомните о возможности управлять бытовой техникой с помощью КПК или универсального пульта ДУ.

Кстати, посмотрите внимательнее на пульт ДУ или на экран программы удаленного контроля. Что вы видите - кнопки? Или кнопки, сгруппированные по функциональному признаку? Да, именно так: кнопки, переключающие каналы, расположены отдельно, рядом - *группа* кнопок, отвечающих за регулировку громкости звука, рядом - *группа* программируемых кнопок, и т. д. В принципе, можно сказать, что пульт реализует не один, а несколько интерфейсов - по числу функциональных групп кнопок. Впрочем, это уже *формализм*: мы просто хотели проиллюстрировать слова "логическая *группа*" в определении интерфейса.

Однако *интерфейс* - это не только и не столько *группа* операций объекта. *Интерфейс* отражает внешние проявления объекта, показывает, каким образом

осуществляется взаимодействие с ним, скрывая остальные детали, не имеющие отношения к процессу взаимодействия.

Интерфейс всегда реализуется некоторым классом, который в таком случае называют классом, *поддерживающим интерфейс*. Как мы уже говорили ранее, один и тот же *объект* может иметь несколько интерфейсов. Это означает, что *класс* этого объекта реализует все *операции* этих интерфейсов. К данному моменту в голове читателя может созреть вопрос: "Мы же, вроде бы, говорили о классах и объектах, а теперь вдруг перешли на интерфейсы. Да и вообще, используются ли они в практике программирования или являются просто изящной теоретической конструкцией?". Ответ на этот вопрос прост: многие из существующих технологий программирования (например, *COM*, *CORBA*, *Java Beans*) не только активно используют *механизм интерфейсов*, но и, по сути, полностью основаны на нем.

Что ж, наверное, пришло время поговорить о том, как *интерфейс* изображается на диаграммах. Изображаться он может несколькими способами. Первый и самый простой из них - это *класс* со стереотипом <<interface>> (рис. 3):

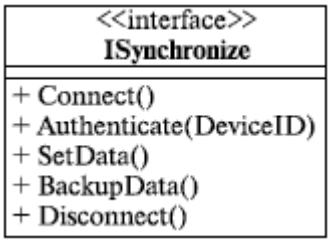


Рис. 3.

Этот способ хорош, если нужно показать, какие именно *операции* предоставляет *интерфейс*. Если же такие подробности в данный момент не важны, предоставляемый *интерфейс* изображают в виде кружочка или, как говорят, "леденца" (lollipop) (рис. 4):

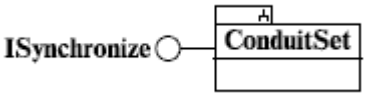


Рис. 4.

Обратите внимание на маленький значок на закладке папки ConduitSet. Это обозначение подсистемы, мы могли бы не рисовать его, а просто использовать стереотип <<subsystem>>. Впрочем, об этом мы еще поговорим.

И наконец, еще один способ изображения интерфейса. Он не является альтернативой описанным ранее способам, а используется для изображения интерфейсов, **требующихся** объекту для выполнения его работы. Обозначается он очень простым и логичным символом. Впрочем, судите сами (рис. 5):

Рис. 5.

Наблюдательный читатель уже, наверное, заметил, как логически совмещаются символы предоставляемого и требуемого интерфейсов.

Действительно, на диаграммах довольно часто можно увидеть такую картинку (рис. 6):

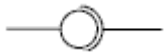


Рис. 6.

Да, кстати, вы заметили, что названия интерфейсов начинаются с буквы I? Эта традиция пошла из языка *Java*, и, как показывает практика, она весьма облегчает жизнь, если нужно, например, быстро разобраться в сложной диаграмме, составленной другим человеком.

Всегда ли нужно создавать новые классы?

Начнем с вопроса, казалось бы, не имеющего никакого отношения к рассматриваемому вопросу, а именно - всегда ли нужно создавать новый *класс* для каждой новой задачи? *Правильный ответ*, конечно же, "нет". Это было бы странно и неэффективно. "Фишка" состоит в том, что мы можем использовать уже существующие классы, адаптируя их функциональность для выполнения новых задач. Таким образом появляется возможность не создавать систему классов с нуля, а задействовать уже имеющиеся решения, которые были созданы ранее, при работе над предыдущими проектами. Впрочем, наше *высказывание* о странности и неэффективности создания новых классов не является истиной в последней инстанции. Могут быть ситуации, когда существующие классы по каким-либо причинам не устраивают архитектора, и тогда требуется создать новый *класс*. Следует, однако, избегать ситуаций, когда созданный *класс* (а точнее, его набор операций и атрибутов) практически повторяет существующий, лишь незначительно отличаясь от него. Все-таки лучше не изобретать велосипед и стараться создавать классы на основе уже существующих, и только если подходящих классов не нашлось - создавать свои, которые, в свою *очередь*, могут (и должны!) служить основой для других классов. Мы уже не говорим о том, что создание классов предполагает значительный объем усилий по кодированию и тестированию. В общем случае, сказанное выше можно проиллюстрировать такой диаграммой (рис. 7):

Рис. 7.

В *дополнение* можно назвать несколько причин, почему стоит использовать уже существующие классы:

Во-первых, идя этим путем, мы пользуемся плодами ранее принятых решений. Действительно, если когда-то мы уже решили некоторую проблему, зачем начинать все "с нуля", повторяя уже однажды проделанные действия?

Во-вторых, таким образом мы делаем решение мобильным и расширяемым. Используя уже существующие классы и создавая на их основе новые, мы можем развивать решение практически неограниченно, добавляя лишь необходимые нам в данный момент детали - атрибуты и *операции*.

В-третьих, существующие классы, как правило, хорошо отлажены и показали себя в работе. Разработчику не надо тратить время на *кодирование*, отладку, тестирование и т. д., - мы работаем с хорошо отлаженным и проверенным временем кодом, который зарекомендовал себя в других проектах и в котором уже выявлено и исправлено большинство ошибок.

А теперь внимание - мы много говорили о том, что нужно создавать классы на основе уже существующих, но так и не сказали ни слова о том, как это сделать. Пришло время внести *ясность* в этот вопрос. Тем самым мы подбираемся к понятию **обобщения** или **генерализации**, которое играет очень важную роль в *ООП*, являясь одним из его базовых принципов. **Обобщение** - это *отношение* между более общей сущностью, называемой *суперклассом*, и ее конкретным воплощением, называемым *подклассом*. Иногда *обобщение* называют отношениями типа "является", имея в виду, что одни сущности (например, круг, квадрат, треугольник) являются воплощением более общей сущности (например, класса "*геометрическая фигура*"). При этом все атрибуты и *операции* суперкласса независимо от *модификаторов видимости* входят в состав подкласса.

Обобщение (или, как часто говорят, *наследование*) на диаграммах обозначается очень просто - незакрашенной треугольной стрелкой, направленной на *суперкласс* (рис. 8).

Для того чтобы научиться эффективно моделировать *наследование*, обратимся к классикам, а именно к Г. Бучу. Он советует проводить эту процедуру в такой последовательности:

1. Найдите атрибуты, операции и обязанности, общие для двух или более классов из данной совокупности. Это позволит избежать ненужного дублирования структуры и функциональности объектов.
2. Вынесите эти элементы в некоторый общий суперкласс, а если такого не существует, то создайте новый класс.
3. Отметьте в модели, что подклассы наследуются от суперкласса, установив между ними отношение обобщения.

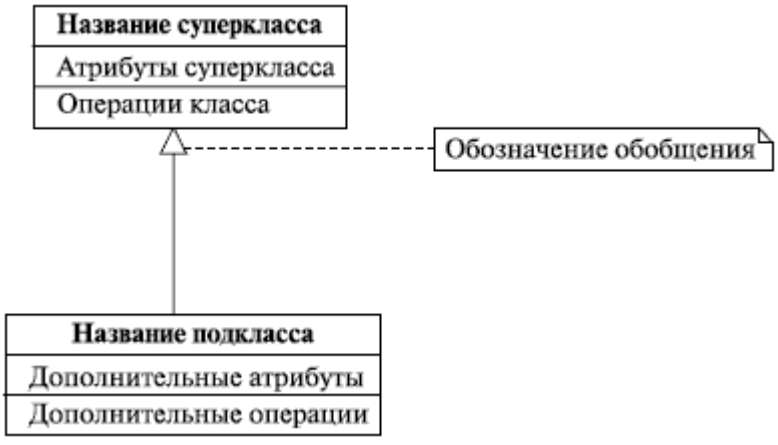


Рис. 8.

А вот и пример применения этого подхода (рис. 9):

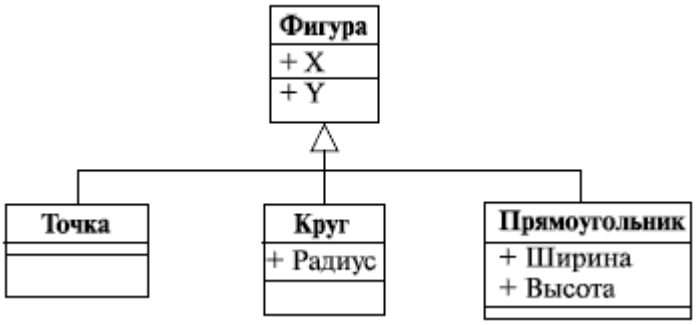


Рис. 9.

На первый взгляд, кажется странным, что *класс* "точка" не имеет никаких атрибутов, а круг имеет только *радиус*. С прямоугольником, вроде бы, все понятно - ширина и *высота*, но вот только где он расположен в пространстве, этот *прямоугольник*? Давайте попробуем следовать советам Буча. Итак, положение всех трех фигур можно однозначно определить с помощью пары чисел. Для точки - это вообще единственные ее характеристики, для круга и прямоугольника - их центры (под центром прямоугольника мы понимаем точку пересечения его диагоналей). Вот они, общие атрибуты! Таким образом, мы создали *суперкласс* "Фигура", имеющий два атрибута - *координаты* центра. Все остальные классы на этой диаграмме связаны с классом "Фигура" отношением обобщения, т. е. в них нужно доопределить только "недостающие" атрибуты - *радиус*, ширину и высоту. Атрибуты, описывающие *координаты* центра, эти классы имеют изначально как потомки класса "Фигура" - они их наследуют. Заметим, что *операции* классов мы тут не рассматриваем: понятно, что с ними была бы та же история.

Так, с наследованием вроде бы разобрались. Пришло время для маленькой провокации с нашей стороны. Классы-потомки ведь наследуют атрибуты и *операции* суперкласса? Таким образом, они могут наследовать и их интерфейсы - то есть объекты абсолютно разной природы могут иметь один и тот же *интерфейс*! Так как же тогда определить, какого же все-таки класса *объект*? Да и нужно ли это вообще?

Действительно, объекты разной природы (или говоря проще, разных классов) могут поддерживать один и тот же *интерфейс* именно так, как того ожидает *пользователь*. Примером тому может служить рассмотренная выше *диаграмма* с геометрическими фигурами. Все рассмотренные фигуры имеют, например, операцию рисования на экране. С точки зрения пользователя в каждом случае это одно и то же действие. Однако реализованы эти *операции* по-разному - ведь процедура изображения прямоугольника сильно отличается от подобной процедуры для круга. Но для пользователя это неважно: ведь *сигнатура*-то одна и та же! А возможно это благодаря еще одному из основных принципов *ООП* - **полиморфизму**. Как мы только что упомянули, работа механизма полиморфизма основана на совпадении сигнатуры метода, объявленного в интерфейсе, и сигнатуры самого метода. Методы внутри классов-потомков могут быть (и наверняка будут!) переопределены, их реализации будут различными, а сигнатуры останутся неизменными. Таким образом (и в этом легко ощутить мощь *ООП*), выполняя одни и те же *операции*, разные объекты могут вести себя по-разному.

Полиморфизм является основой для реализации *механизма интерфейсов* в языках программирования. Вот, кстати, и ответ на вопрос, какого класса *объект*: как только *пользователь* обращается к некоторой *операции* через *интерфейс*, определяется фактический *класс* объекта и вызывается соответствующая *операция класса*. Примеры полиморфизма можно увидеть в самых обыденных вещах, которыми мы пользуемся в повседневной жизни. Оглянитесь вокруг - мир построен по *ООП, Матрица* работает! Например, всем привычная кредитная карточка, является интерфейсом для доступа к банковскому счету через банкомат (и не только), одинаково работает в любой стране, вот только ведет себя чуть-чуть по-разному, т. к. банкомат выдает деньги в местной валюте.

Согласны, пример не очень корректный, но зато очень наглядный! Думаем, понаблюдав за окружающим миром, читатель сам сможет привести массу примеров полиморфизма.

Инкапсуляция, *наследование* и *полиморфизм*, с которыми мы только что познакомились, являются теми самыми тремя китами, на которых держится *ООП*. Если вы поняли суть этих базовых принципов и осознали их истинную мощь, вы прошли большую часть пути, ведущего к полному овладению *ООП* как наиболее адекватной методикой описания (так и тянет сказать "проектирования") окружающего нас мира.

Отношения между классами

Ни один из объектов окружающего нас мира не существует сам по себе. Птицы летают потому, что есть воздух, на который опираются их крылья. Каждый из нас связан с массой других людей разнообразными родственными, профессиональными и другими связями, предполагающими различные типы отношений. Точно так же и классы связаны между собой. И чтобы в полной мере овладеть *ООП*, нам необходимо понять суть этих отношений и научиться их идентифицировать.

Мы сказали, что объекты находятся в определенных отношениях друг с другом. Один из типов таких отношений - это **зависимость**. Думаем, суть такого отношения понятна уже из его названия - зависимость возникает тогда, когда реализация класса одного объекта зависит от спецификации операций класса другого объекта. И если изменится спецификация операций этого класса, нам неминуемо придется вносить изменения и в зависимый *класс*. Приведем простой пример, опять-таки взятый из нашей повседневности. Иногда к нам в руки попадают видеофайлы, воспроизвести которые "с лету" не удается. Почему? Правильно, потому что на компьютере не установлены соответствующие кодеки. То есть операция "Воспроизведение", реализуемая программой-медиаплеером, зависит от *операции* "Декомпрессия", реализуемой кодеком. Если спецификация *операции* "Декомпрессия" изменится, придется менять код медиаплеера, иначе он просто не сможет работать с каким-то кодеком и, в лучшем случае, завершит свою работу с ошибкой. А вот так зависимость между классами изображается в *UML* (рис. 10):

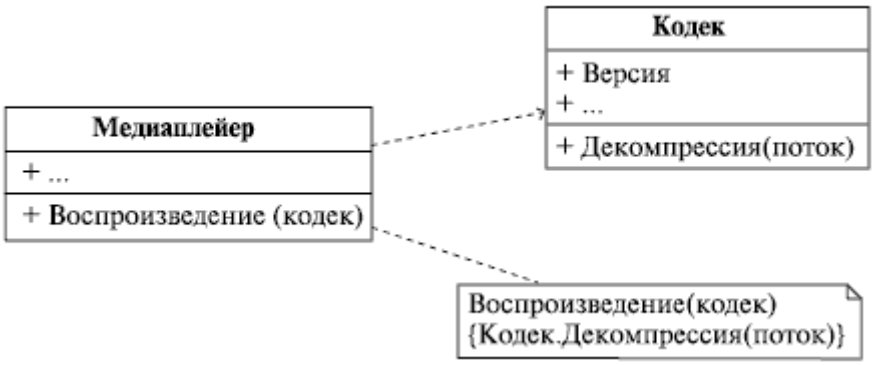


Рис. 10.

Стоит отметить, что зависимости на диаграммах изображают далеко не всегда, а только в тех случаях, когда их *отображение* является важным для понимания модели. Часто зависимости лишь подразумеваются, т. к. логически следуют из природы классов.

Другой вид отношений между объектами - это **ассоциация**. Это просто *связь* между объектами, по которой можно между ними перемещаться. *Ассоциация* может иметь имя, показывающее природу отношений между объектами, при этом в имени может указываться *направление* чтения связи при помощи треугольного маркера. Однонаправленная *ассоциация* может изображаться стрелкой. Проиллюстрируем сказанное примерами (рис. 11):

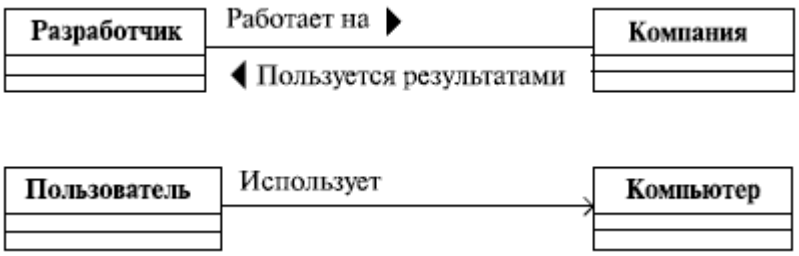


Рис. 11.

Кроме направления ассоциации, мы можем указать на диаграмме *роли*, которые каждый *класс* играет в данном отношении, и *кратность*, то есть количество объектов, связанных отношением (рис. 12):

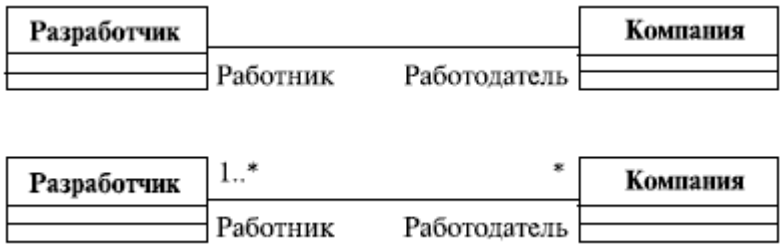


Рис. 12.

И насчет ролей, и насчет кратности на этой диаграмме все понятно - человек может вообще не работать, работать в одной или более компаниях, а вот компании в любом случае нужен хотя бы один сотрудник. Кстати, о кратности. *Ассоциация* может объединять три и более класса. В этом случае она называется **n-арной** и изображается ромбом на пересечении линий, как показано на этой диаграмме, позаимствованной нами из Zicom Mentor (рис. 13):

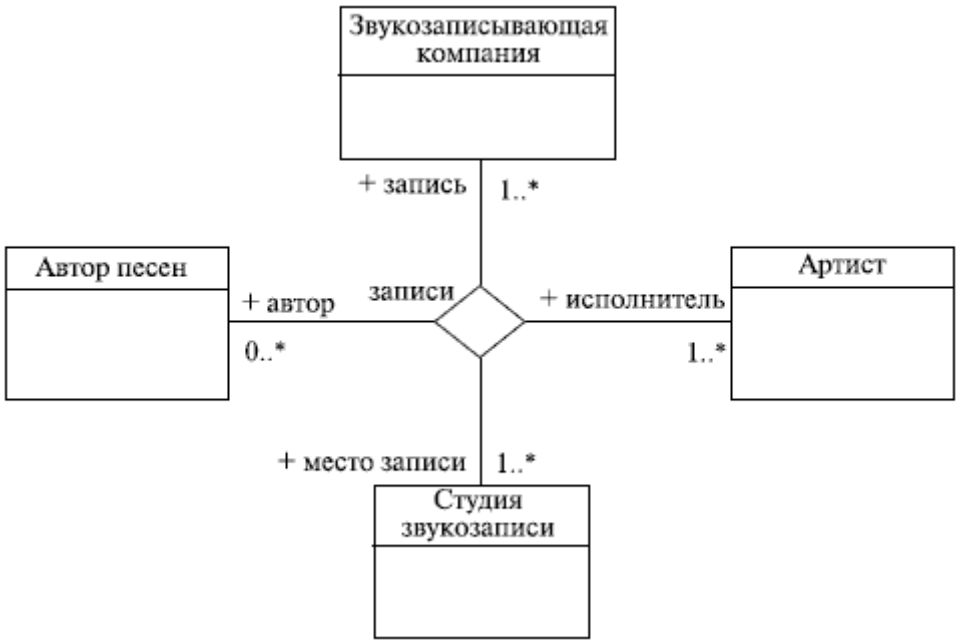


Рис. 13.

Ранее мы говорили, что *ассоциация* - это "просто *связь*" между объектами. На самом деле, в реальности связи бывают "просто связями" крайне редко. Обычно при ближайшем рассмотрении под ассоциацией понимается более сложное *отношение* между классами, например, *связь* типа "часть-целое". Такой вид ассоциации называется **ассоциацией с агрегированием**. В этом случае один *класс* имеет более высокий статус (целое) и состоит из низших по статусу классов (частей). При этом выделяют простое и композитное *агрегирование* и говорят о собственно **агрегации** и **композиции**. Простая *агрегация* предполагает, что части, отделенные от целого, могут продолжать свое существование независимо от него. Под композитным же агрегированием понимается ситуация, когда целое владеет своими частями и их время жизни соответствует времени жизни целого, т. е. независимо от целого части существовать не могут. Примеры этих видов ассоциаций и их обозначений в *UML* можно увидеть на следующей диаграмме (рис. 14).

Рис. 14.

Примеры, как нам кажется, очень простые и понятные. *Винчестер* можно вынуть из компьютера и установить в новый *компьютер* или в USB-карман, т. е. существование жесткого диска с разборкой системного блока не заканчивается. А вот кнопки без окон обычно существовать не могут - с закрытием окна кнопки также исчезают.

И, наконец, еще одна важная вещь, касающаяся ассоциации. В отношении между двумя классами сама *ассоциация* тоже может иметь свойства и, следовательно, тоже может быть представлена в виде класса. Пример прост (рис. 15):

Рис. 15.

Действительно, перед началом трудовых отношений работник и работодатель подписывают между собой контракт, который имеет такие атрибуты, как, например, описание *работ*, сроки их выполнения, порядок оплаты и т. д.

А вот более сложный, но, опять-таки, взятый из реальной жизни пример моделирования отношений между классами, позаимствованный нами из Zicom Mentor (рис. 16):

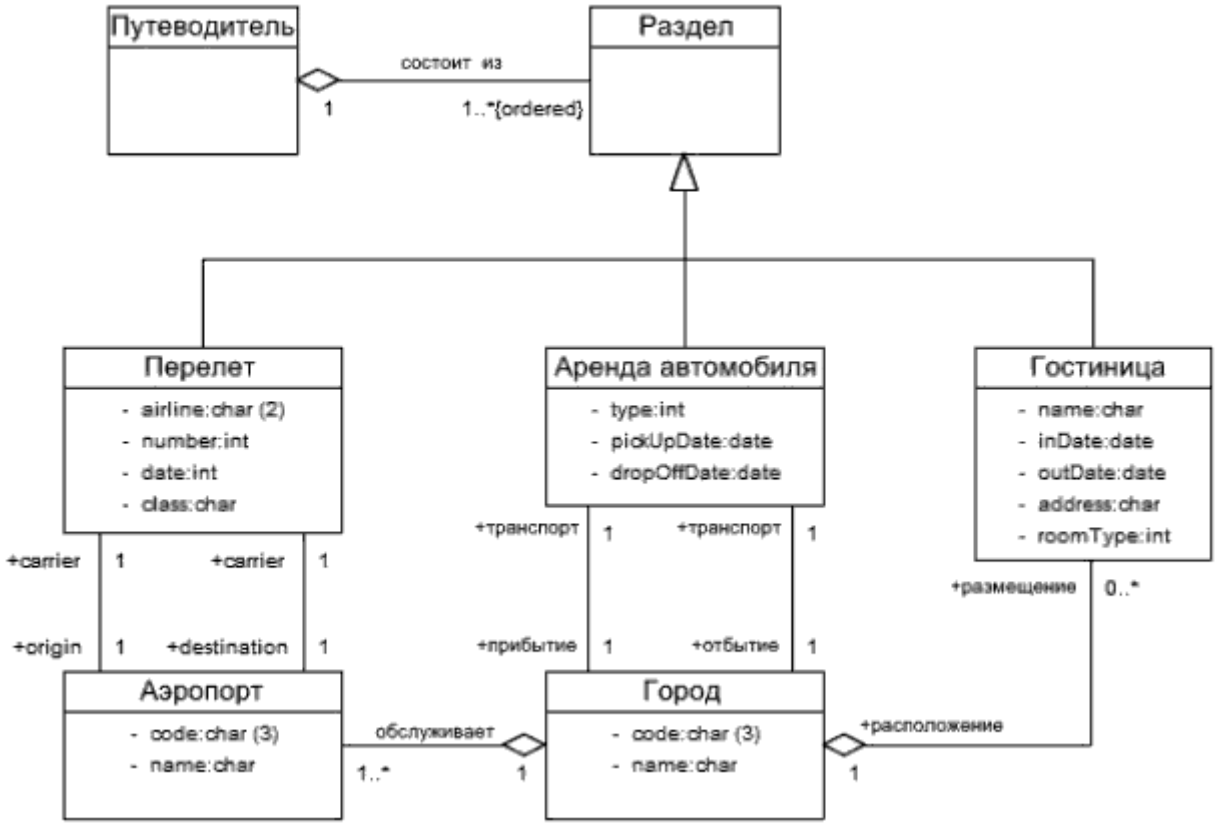


Рис. 16.

Практическое задание. Используя предметные области (см. список ниже), выполните следующие действия:

1. Определите, какие объекты предметной области будут описаны на диаграмме классов.
2. В графическом редакторе (удобном лично Вам) создайте классы и объекты, входящие в них, с использованием стандартов языка UML.

Определите связи между классами и входящими в них объектами и нанесите их на Вашу диаграмму.

Лабораторная работа №11. Построение UML-диаграмм использования (Use Case)

Теоретический материал.

В языке UML для формализации функциональных требований *применяются диаграммы использования*.

Диаграмму вариантов использования есть смысл строить во время *изучения технического задания*, она состоит из графической диаграммы, описывающей *действующие лица и прецеденты*, а также спецификации, представляющего собой текстовое описание конкретных *последовательностей действий (потока событий)*, которые выполняет пользователь при работе с системой. Спецификация затем станет **основой для тестирования и документации,** а на следующих этапах проектирования она дополняется и оформляется в виде диаграммы (в рамках ICONIX используется диаграмма последовательности, но в UML для этого имеются также диаграммы деятельности). Кроме того, use-case диаграмма достаточно проста, чтобы ее мог понять заказчик, следовательно вы **можете использовать ее для согласования ТЗ** (ведь диаграмма описывает функциональные требования к системе).

На диаграмме использования изображаются:

- акторы — группы лиц или систем, взаимодействующих с нашей системой;
- варианты использования (прецеденты) — сервисы, которые наша система предоставляет акторам;
- комментарии;
- отношения между элементами диаграммы.

На мой взгляд, наиболее правильный порядок построения диаграммы следующий:

1. выделить группы действующих лиц (работающих с системой по-разному, часто из-за различных прав доступа);
2. идентифицировать как можно больше вариантов использования (процессов, которые могут выполнять пользователи). При этом не следует делить процессы слишком мелко, нужно выбирать лишь те, которые дадут пользователю **значимый результат.** Например, кассир может «продать товар» (это будет являться прецедентом), однако «ввод штрих-кода товара для получения цены» самостоятельным прецедентом не является;
3. дополнить прецеденты словесным описанием (сценарием):
    - для каждого прецедента создать разделы: «главная последовательность» и «альтернативные последовательности»;
    - при составлении сценария нужно упорно задавать заказчику вопросы «что происходит?», «что дальше?», «что еще может происходить?» и записывать ответы на них.

Сценарии являются очень важной частью диаграмм использования, хотя их формат и не регламентирован. Ряд авторов предлагает использовать *псевдокод* для представления сценария и даже сразу строить *диаграммы деятельности* или *взаимодействия*, но на мой взгляд, наиболее предпочтительным вариантом на этапе построения *use-case диаграмм* является текстовый, описывающий систему с точки зрения пользователя (т.к. именно этот формат будет наиболее понятен заказчику, с которым вам предстоит *согласовывать техническое задание*).

Рассмотрим *разработку диаграмм вариантов использования на примере* — пусть заказчик дал нам следующее техническое задание:

Цель — развитие у детей математических навыков.
Платформа: Linux, Windows, Android.
Функциональность:

для учеников:

выбор подготовленного учителем блока заданий;

выполнение заданий;

для учителя:

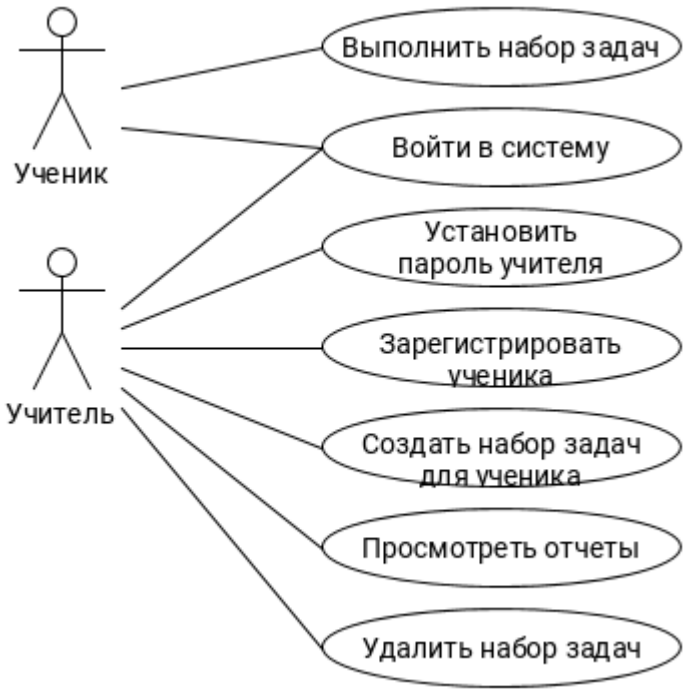
подготовка для учеников блоков заданий;

добавление в систему ученика;

просмотр отчетов.

При первом запуске система должна позволять ввести пароль учителя. Задания представляют собой математические задачи на сложение, вычитание, умножение и деление. В блоке задач могут быть задачи различных типов (указывается количество). Помимо ввода типа выполняемой в примере операции необходимо указывать допустимые диапазоны чисел (или даже отдельные числа, т.к. при изучении таблицы умножения часто сначала учат умножение на 2, затем на 5, а только потом все остальное). Кроме того, для операции вычитания необходимо иметь возможность установить вычитаемое меньше уменьшаемого (т.к. в противном случае результат будет отрицательным, а отрицательные числа в школе проходят гораздо позже).

Очевидно, несмотря на то, что заказчик очень подробно описал некоторые детали, мы не можем не только приступить к реализации задачи, но даже приблизительно оценить стоимость и сроки выполнения. Из такого задания не понятно, например, что должны содержать отчеты. Однако, мы сразу можем выделить две группы пользователей и несколько видов их деятельности.



Пример диаграммы использования

Сплошные линии на диаграмме представляют собой отношения ассоциации, отражающие возможность использования актором прецедента. После того, как определен набор вариантов использования, можно приступать к составлению сценариев. Сценарии должны описываться с точки зрения пользователя, при этом важно описывать взаимодействие пользователя с элементами интерфейса. Так, например сценарий прецедента регистрации ученика мог бы выглядеть следующим образом:

Название прецедента: регистрация ученика

Действующее лицо: учитель

Цель: добавить ученика в систему, получив его пароль

Предусловия: учитель осуществил вход в систему

Главная последовательность:

учитель выбирает в *главном меню* пункт «*добавить ученика*«;

система показывает учителю *окно добавления ученика,* содержащее *поля для ввода логина и пароля*, а также *кнопки «далее» и «назад»*;

учитель вводит желаемый логин и пароль ученика, нажимает кнопку *«далее»*;

система добавляет ученика;

учителю открывается *главное меню* и в течении 5 секунд выводится уведомление о том, что ученик был добавлен успешно.

Альтернативная последовательность (возврат в главное меню без добавления ученика):

учитель выбирает в *главном меню* пункт «*добавить ученика*«;

система показывает учителю *окно добавления ученика,* содержащее *поля для ввода логина и пароля*, а также *кнопки «далее» и «назад»*;

учитель нажимает кнопку *«назад»*;

учителю открывается *главное меню (*при этом данные, введенные в формы *окна добавления ученика* не сохраняются).

Альтернативная последовательность (добавление ученика, уже имеющегося в системе):

учитель выбирает в *главном меню* пункт «*добавить ученика*«;

система показывает учителю *окно добавления ученика,* содержащее *поля для ввода логина и пароля*, а также *кнопки «далее» и «назад»*;

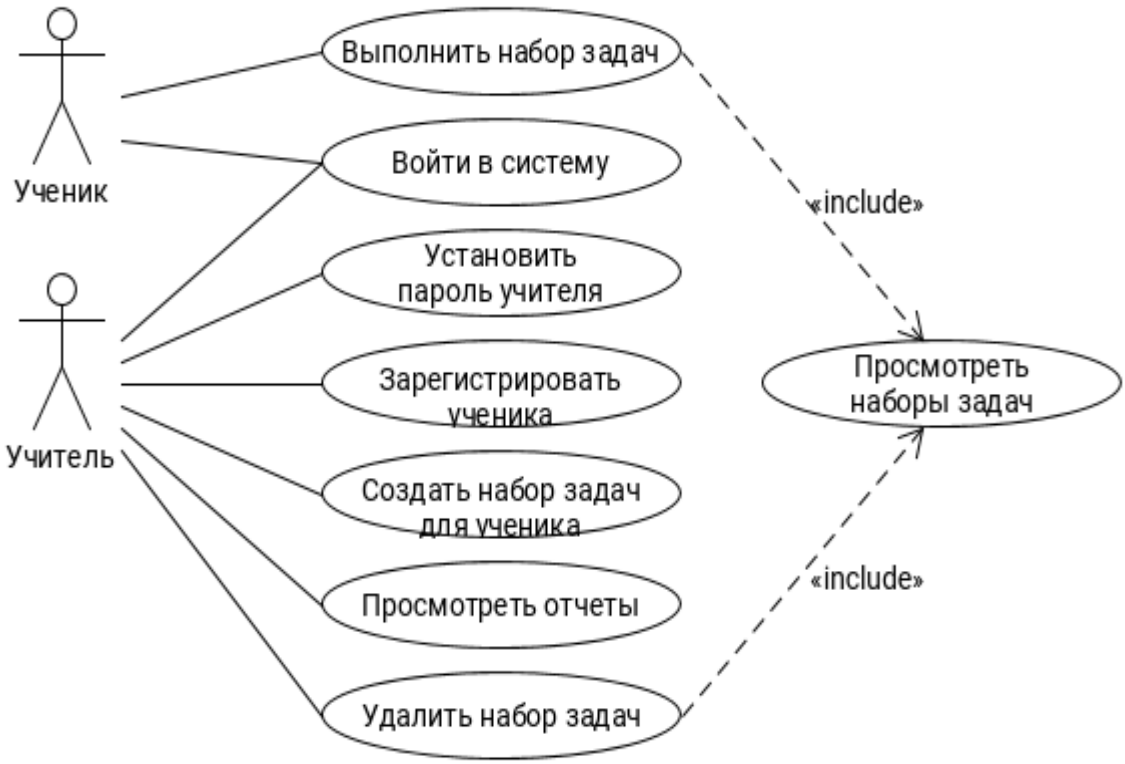
учитель вводит желаемый логин и пароль ученика, нажимает кнопку *«далее»*;

учителю в течении 5 секунд отображается уведомление о том, что запрашиваемый логин занят.

Аналогичным образом должны быть прописаны все прецеденты, изображенные на диаграмме. Составлять сценарии нужно достаточно упорно чтобы описать все возможные варианты действий пользователя в системе. Заказчик может делать это с большим удовольствием, а программист за счет этого раньше узнает возможные пожелания заказчика (так из приведенного сценария он мог бы выяснить, что программа должна отображать всплывающие уведомления).

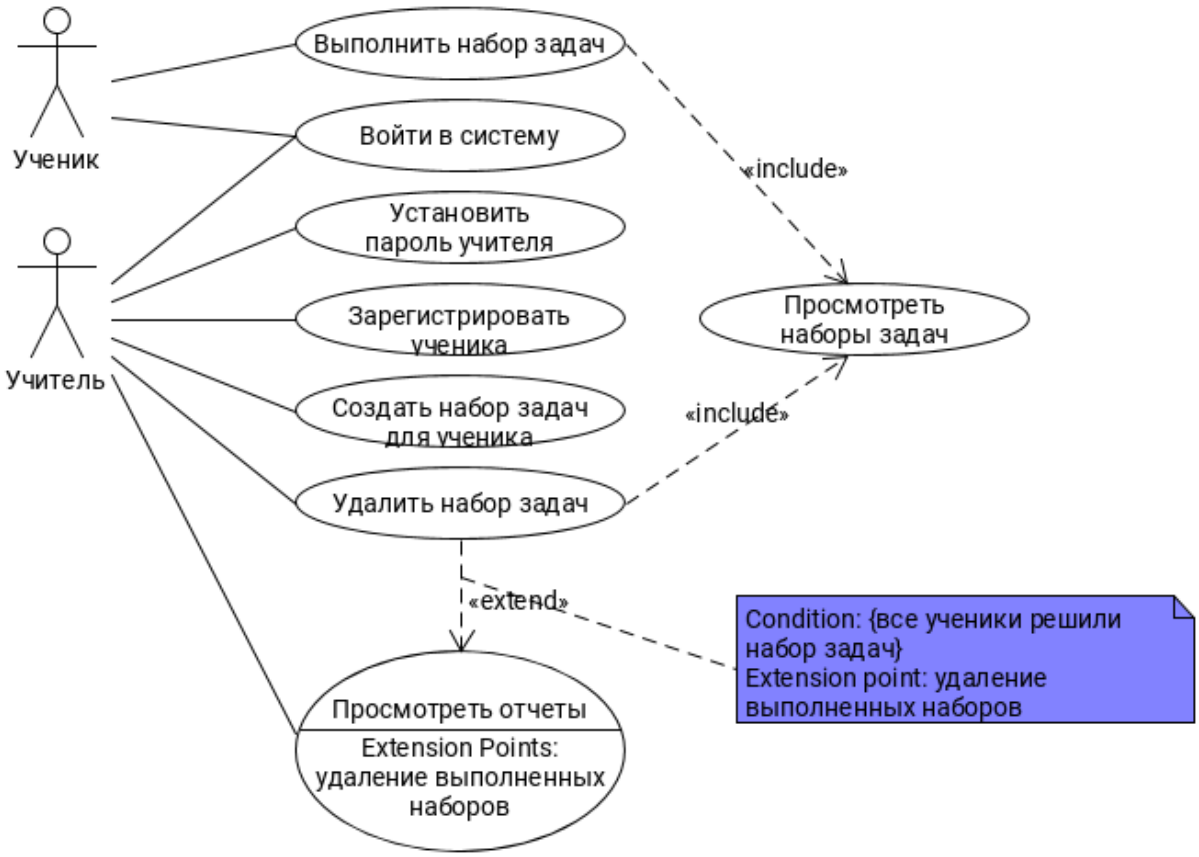
Несмотря на простоту приведенного сценария, в его последовательностях можно найти дублирование, если оно имеет место в ваших сценариях — вы можете выделить некоторые фрагменты описания в отдельные прецеденты (которые могут быть как самостоятельными, так и являться лишь частью других вариантов использования). При этом между прецедентами появится либо отношение **расширения (extend),** либо **включения (include)**, которые отображаются на диаграммах (в *UML* также существует отношение обобщения, а в *OML* — вызова и предшествования).

Отношение включения указывает на то, что поведение одного прецедента включается в некоторой точке в другой прецедент в качестве составного компонента. Особенности включения заключаются в том, что включаемый прецедент должен быть обязательным для дополняемого (включение должно быть безусловным, а дополняемый вариант использования без включения не сможет выполняться), т.е. это это отношение задает очень сильную связь. Например, если мы хотим изобразить на диаграмме тот факт, что удаление набора задач учителем и выполнение задач учеником не должно происходить без ***обязательного*** просмотра всех наборов задач — то нам нужно использовать отношение включения:



Отношение включения на диаграмме использования

Отношение расширения отражает ***возможное*** присоединение одного варианта использования к другому в некоторой точке (*точке расширения*). При этом подчеркивается то, что расширяющий вариант использования выполняется лишь при определенных условиях и не является обязательным для выполнения основного прецедента. На диаграмме такой вид отношения изображается стрелкой, направленной к расширяемому прецеденту, в отдельном разделе которого *может быть описана* ***точка расширения***, а ***условия расширения*** *могут быть приведены* в комментарии с *ключевым словом* ***Condition***. Таким образом, расширение позволяет моделировать *необязательное поведение системы*, которое является *условным* и *не изменяет поведение* основного прецедента. Например отношение расширения нужно применить, если по техническому заданию *требуется возможность* удаления набора задач в *прецеденте просмотра отчетов* при условии, что все ученики решили этот набор.



Отношение расширения на диаграмме использования

Таким образом можно показать, что у учителя появляется возможность (но не обязанность) удалить набор задач при просмотре отчетов если все ученики выполнили этот набор.

На последней диаграмме используется символ комментария для задания условий расширения, при этом комментарий связывается пунктирной линией с отношением расширения, т.к. относится к нему. В ряде публикаций по UML и ICONIX предлагается описывать с помощью комментариев на диаграмме прецедентов:

- нефункциональные требования к системе (при этом используется *стереотип <<requirement>>*);
- сценарии вариантов использования (связывая комментарий с соответствующим прецедентом);
- детали реализации и другие выводы, к которым разработчики пришли в процессе обсуждения задачи (не все с этим согласны, т.к. use-case диаграмма показывается заказчику, которому не нужны детали).

Наиболее **типичными ошибками** при построении этого вида диаграмм являются:

- неправильное использование отношений расширения и включения, в том числе попытки использовать диаграммы для функциональной декомпозиции системы**.** Возникает из-за непонимания различий между этими двумя видами

отношений и того, что use-case диаграмма должна выражать лишь требования к системе, а не детали ее реализации;

- разработка диаграммы с точки зрения программиста, а не пользователя. В сценариях должны использоваться названия элементов управления (видимые пользователю), но нежелательно изображать детали реализации (такие как менеджер событий), не понятные заказчику;
- не достаточная проработка сценариев:
    - отсутствие или недостаточное количество альтернативных последовательностей, в которых должен быть учтен, в том числе, ввод некорректных данных в систему;
    - описание действий пользователя без указания конкретных элементов интерфейса системы и отсутствие описаний реакции системы в сценариях.

Важно, что *процесс ICONIX* является итеративным, поэтому если вы допустите неточности на этапе разработке диаграмм использования — их можно будет найти и исправить на следующих этапах (в частности, пропущенные объекты могут быть выделены при работе над диаграммами робастности, а сценарии детально проработаны при построении диаграмм последовательности).

Стоит отметить, что нет единого мнения по поводу использования в текстах сценария условных операторов или циклов. Ряд аналитиков считают, что наличие слов типа «если» в сценарии является ошибкой, которая исправляется добавлением альтернативной последовательности. Другие — допускают использование 1-2 ветвлений в сценарии, при этом отмечают, что такой подход улучшает читабельность сценариев.

Если следовать всем приведенным правилам составления диаграмм вариантов использования, с их помощью можно достаточно подробно ***проработать техническое задание*** чтобы ***оценить сроки*** и стоимость его выполнения, описать конкретные сценарии взаимодействия с системой, которые лягут в ***основу тестов и документации***, и ***согласовать*** всё это с заказчиком.

Практическое задание. Используя предметные области (см. список ниже), выполните следующие действия:

1. Определите, какие действующие лица (экторы) можно выделить для вашей предметной области.
2. В графическом редакторе (удобном лично Вам) создайте экторы и прецеденты, с ними связанные, с использованием стандартов языка UML.

Определите отношения между экторами и прецедентами и нанесите их на Вашу диаграмму.

Лабораторная работа №12. Построение UML-диаграмм компонентов

Теоретический материал.

Диаграмма компонентов и особенности ее построения

Все рассмотренные ранее диаграммы отражали концептуальные и логические аспекты построения модели системы. Особенность логического представления заключается в том, что оно оперирует понятиями, которые не имеют материального воплощения. Другими словами, различные элементы логического представления, такие как классы, ассоциации, состояния, сообщения, не существуют материально или физически. Они лишь отражают понимание статической структуры той или иной системы или динамические аспекты ее поведения.

Для создания конкретной физической системы необходимо реализовать все элементы логического представления в конкретные материальные сущности. Для описания таких реальных сущностей предназначен другой аспект модельного представления, а именно – физическое *представление* модели. В контексте языка *UML* это означает совокупность связанных физических сущностей, включая программное и *аппаратное обеспечение*, а также персонал, которые организованы для выполнения специальных задач.

Физическая система (*physical system*) — реально существующий прототип модели системы.

С тем чтобы пояснить отличие логического и физического представлений, необходимо в общих чертах рассмотреть процесс разработки программной системы. Ее исходным логическим представлением могут служить структурные схемы алгоритмов и процедур, описания интерфейсов и *концептуальные схемы баз* данных. Однако для реализации этой системы необходимо разработать исходный текст программы на языке программирования. При этом уже в тексте программы предполагается организация программного кода, определяемая синтаксисом языка программирования и предполагающая *разбиение* исходного кода на отдельные модули.

Однако исходные тексты программы еще не являются окончательной реализацией проекта, хотя и служат фрагментом его физического представления. Программная система может считаться реализованной в том случае, когда она будет способна выполнять функции своего целевого предназначения. А это возможно, только если программный код системы будет реализован в форме исполняемых модулей, библиотек классов и процедур, стандартных графических интерфейсов, файлов баз данных. Именно эти компоненты являются базовыми элементами физического представления системы в нотации языка *UML*.

Полный проект программной системы представляет собой совокупность моделей логического и физического представлений, которые должны быть согласованы между собой. В языке *UML* для физического представления моделей систем используются так называемые диаграммы реализации, которые включают в себя две отдельные *канонические диаграммы*: диаграмму компонентов и *диаграмму развертывания*.

Диаграмма компонентов, в отличие от ранее рассмотренных диаграмм, описывает особенности физического представления системы. *Диаграмма компонентов* позволяет определить архитектуру разрабатываемой системы, установив зависимости между программными *компонентами*, в роли которых может выступать исходный, бинарный и *исполняемый код*. Во многих средах разработки *модуль* или *компонент* соответствует файлу. Пунктирные стрелки, соединяющие *модули*, показывают отношения взаимозависимости, аналогичные тем, которые имеют *место* при компиляции исходных текстов программ. Основными графическими элементами диаграммы *компонентов* являются *компоненты*, *интерфейсы* и зависимости между ними.

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 отношений между ними.

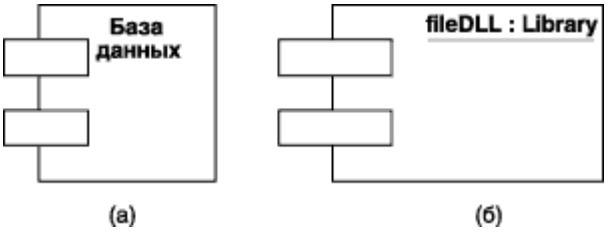
Компоненты

Для представления физических сущностей в языке *UML* применяется специальный термин – *компонент*.

Компонент (component) — физически существующая часть системы, которая обеспечивает реализацию классов и отношений, а также функционального поведения моделируемой программной системы.

Компонент предназначен для представления физической организации ассоциированных с ним элементов модели. Дополнительно *компонент* может иметь текстовый стереотип и *помеченные значения*, а некоторые *компоненты* – собственное графическое *представление*. *Компонентом* может быть *исполняемый код* отдельного *модуля*, командные файлы или файлы, содержащие интерпретируемые скрипты.

Компонент служит для общего обозначения элементов физического представления модели и может реализовывать некоторый набор *интерфейсов*. Для графического представления *компонента* используется *специальный символ – прямоугольник* со вставленными слева двумя более мелкими прямоугольниками . Внутри объемлющего прямоугольника записывается имя *компонента* и, возможно, дополнительная *информация*. Этот символ является базовым обозначением *компонента* в языке *UML*.



Графическое изображение компонента

Графическое изображение *компонента* ведет свое происхождение от обозначения *модуля* программы, применявшегося некоторое время для отображения особенностей *инкапсуляции данных* и процедур.

Модуль (module) — часть программной системы, требующая памяти для своего хранения и процессора для исполнения.

В этом случае верхний маленький *прямоугольник* концептуально ассоциировался с данными, которые реализует этот *компонент* (иногда он изображается в форме овала). Нижний маленький *прямоугольник* ассоциировался с операциями или методами, реализуемыми *компонентом*. В простых случаях имена данных и методов записывались явно в маленьких прямоугольниках, однако в языке *UML* они не указываются.

Имя *компонента* подчиняется общим правилам именования элементов модели в языке *UML* и может состоять из любого числа букв, цифр и знаков препинания. Отдельный *компонент* может быть представлен на уровне типа или экземпляра. И хотя его графическое изображение в обоих случаях одинаково, правила записи имени *компонента* несколько отличаются.

Если *компонент* представляется на уровне типа, то записывается только имя типа с заглавной буквы в форме: <Имя типа>. Если же *компонент* представляется на уровне экземпляра, то его имя записывается в форме: <имя *компонента* ‘:' Имя типа>. При этом вся строка имени подчеркивается. Так, в первом случае для *компонента* уровня типов указывается имя типа, а во втором для *компонента* уровня экземпляра – собственное имя *компонента* и имя типа.

Правила именования объектов в языке *UML* требуют подчеркивания имени отдельных экземпляров, но применительно к *компонентам* подчеркивание их имени часто опускают. В этом случае *запись* имени *компонента* со строчной буквы характеризует *компонент* уровня примеров.

В качестве собственных имен *компонентов* принято использовать имена исполняемых файлов, динамических библиотек, Web-страниц, текстовых файлов или файлов справки, файлов баз данных или файлов с исходными текстами программ, файлов скриптов и другие.

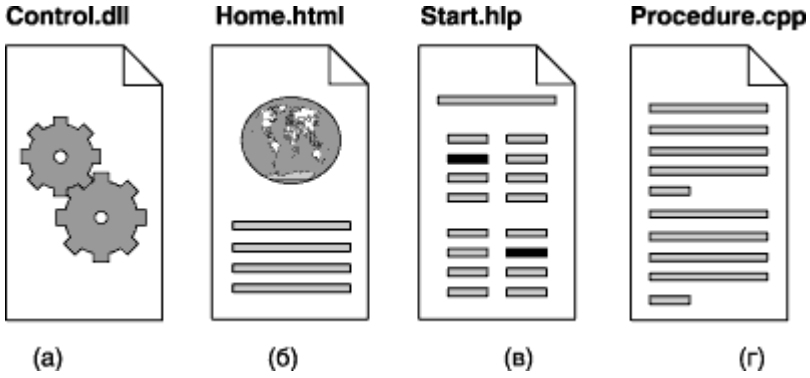
В отдельных случаях к простому имени *компонента* может быть добавлена *информация* об имени объемлющего пакета и о конкретной версии реализации данного *компонента*. Необходимо заметить, что в этом случае номер версии записывается как помеченное *значение* в фигурных скобках. В других случаях символ *компонента* может быть разделен на секции, чтобы явно указать имена реализованных в нем классов или *интерфейсов*. Такое обозначение *компонента* называется **расширенным** .

Поскольку *компонент* как элемент модели может иметь различную физическую реализацию, иногда его изображают в форме специального графического символа, иллюстрирующего

конкретные особенности реализации. Строго говоря, эти дополнительные обозначения не специфицированы в нотации языка *UML*. Однако, удовлетворяя общим механизмам расширения языка *UML*, они упрощают понимание диаграммы *компонентов*, существенно повышая наглядность графического представления.

Для более наглядного изображения *компонентов* были предложены и стали общепринятыми следующие графические стереотипы:

- Во-первых, стереотипы для *компонентов* развертывания, которые обеспечивают непосредственное выполнение системой своих функций. Такими *компонентами* могут быть динамически подключаемые библиотеки, Web-страницы на языке разметки гипертекста и файлы справки.
- Во-вторых, стереотипы для *компонентов* в форме *рабочих продуктов*. Как правило – это файлы с исходными текстами программ



Варианты графического изображения компонентов на диаграмме компонентов.

Эти элементы иногда называют **артефактами** , подчеркивая при этом их законченное информационное содержание, зависящее от конкретной технологии реализации соответствующих *компонентов*. Более того, разработчики могут для этой цели использовать самостоятельные обозначения, поскольку в языке *UML* нет строгой нотации для графического представления артефактов.

Другой способ спецификации различных видов *компонентов* — указание текстового стереотипа *компонента* перед его именем. В языке *UML* для *компонентов* определены следующие стереотипы:

- <<file>> (файл) – определяет наиболее общую разновидность *компонента*, который представляется в виде произвольного физического файла.
- <<executable>> (исполнимый) – определяет разновидность компонента-файла, который является исполнимым файлом и может выполняться на компьютерной платформе.
- <<document>> (документ) – определяет разновидность компонента-файла, который представляется в форме документа произвольного содержания, не являющегося исполнимым файлом или файлом с исходным текстом программы.

- <<library>> (библиотека) – определяет разновидность компонента-файла, который представляется в форме динамической или статической библиотеки.
- <<source>> (источник) – определяет разновидность компонента-файла, представляющего собой файл с исходным текстом программы, который после компиляции может быть преобразован в исполнимый файл.
- <<table>> (таблица) – определяет разновидность *компонента*, который представляется в форме таблицы базы данных.

Отдельными разработчиками предлагались собственные графические стереотипы для изображения тех или иных типов *компонентов*, однако, за небольшим исключением они не нашли широкого применения. В свою *очередь* ряд инструментальных CASE-средств также содержат дополнительный набор графических стереотипов для обозначения *компонентов*.

Интерфейсы

Следующим графическим элементом диаграммы *компонентов* являются *интерфейсы*. В общем случае *интерфейс* графически изображается окружностью, которая соединяется с *компонентом* отрезком линии без стрелок (рис. 12.3, а). При этом имя *интерфейса*, которое рекомендуется начинать с заглавной буквы "I", записывается рядом с окружностью. Семантически линия означает реализацию *интерфейса*, а наличие *интерфейсов* у *компонента* означает, что данный *компонент* реализует соответствующий набор *интерфейсов*.

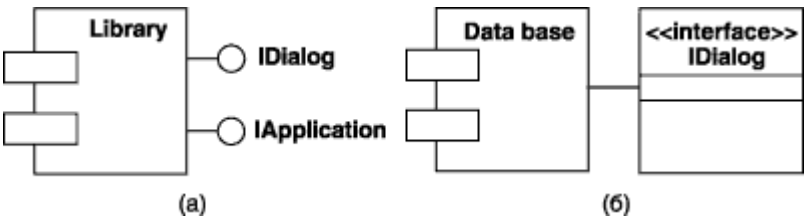


Рис. 12.3. Графическое изображение интерфейсов на диаграмме компонентов.

Кроме того, *интерфейс* на диаграмме *компонентов* может быть изображен в виде прямоугольника класса со стереотипом << *interface*>> и секцией поддерживаемых операций (рис. 12.3, б). Как правило, этот вариант обозначения используется для представления внутренней структуры *интерфейса*.

При разработке программных систем *интерфейсы* обеспечивают не только совместимость различных версий, но и возможность вносить существенные изменения в одни части программы, не изменяя другие . Характер применения *интерфейсов* отдельными *компонента* ми может отличаться.

Различают два способа связи *интерфейса* и *компонента*. Если *компонент* реализует некоторый *интерфейс*, то такой *интерфейс* называют *экспортируемым* или **поддерживаемым**, поскольку этот *компонент* предоставляет его в качестве сервиса другим *компонентам* . Если же *компонент* использует некоторый *интерфейс*, который реализуется другим *компонентом*, то такой *интерфейс* для первого *компонента* называется ***импортируемым*** . Особенность *импортируемого интерфейса* состоит в том, что на диаграмме *компонентов* это *отношение* изображается с помощью зависимости.

Зависимости между компонентами

В общем случае *отношение* зависимости также было рассмотрено ранее. *Отношение* зависимости служит для представления факта наличия специальной формы связи между двумя элементами модели, когда изменение одного элемента модели оказывает влияние или приводит к изменению другого элемента модели. *Отношение* зависимости на диаграмме *компонентов* изображается пунктирной линией со стрелкой, направленной от клиента или зависимого элемента к источнику или независимому элементу модели.

Зависимости могут отражать связи отдельных файлов программной системы на этапе компиляции и генерации объектного кода. В других случаях зависимость может указывать на наличие в *независимом компоненте* описаний классов, которые используются в *зависимом компоненте* для создания соответствующих объектов. Применительно к диаграмме *компонентов* зависимости могут связывать *компоненты* и импортируемые этим *компонентом интерфейсы*, а также различные виды *компонентов* между собой.

В этом случае рисуют стрелку от компонента-клиента к *импортируемому интерфейсу* (рис. 12.4). Наличие такой стрелки означает, что *компонент* не реализует соответствующий *интерфейс*, а использует его в процессе своего выполнения. При этом на этой же диаграмме может присутствовать и другой *компонент*, который реализует этот *интерфейс*. *Отношение* реализации *интерфейса* обозначается на диаграмме *компонентов* обычной линией без стрелки.

Так, например, изображенный ниже фрагмент *диаграммы компонентов* представляет информацию о том, что *компонент* с именем *Control* зависит от *импортируемого интерфейса* IDialog, который, в свою *очередь*, реализуется *компонентом* с именем DataBase . При этом для второго *компонента* этот *интерфейс* является *экспортируемым*. Изобразить *связь* второго *компонента* DataBase с этим *интерфейсом* в форме зависимости нельзя , поскольку этот *компонент* реализует указанный *интерфейс*.

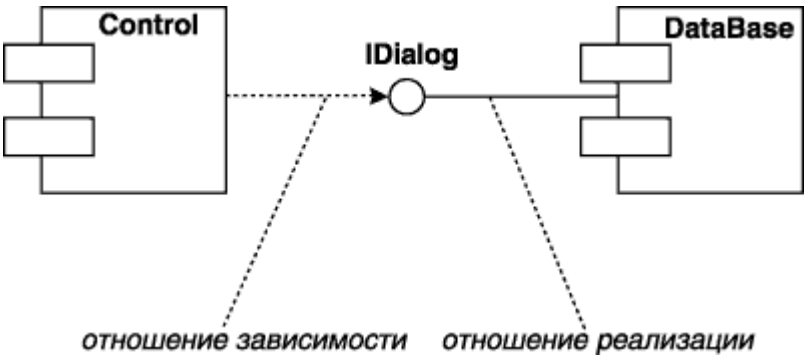


Рис. 12.4. Фрагмент диаграммы компонентов с отношениями зависимости и реализации

Другим случаем отношения зависимости на диаграмме компонентов является *отношение* программного вызова и компиляции между различными видами *компонентов*. Для рассмотренного фрагмента *диаграммы компонентов* (рис. 12.5) наличие подобной зависимости означает, что исполнимый *компонент Control* .exe использует или импортирует некоторую функциональность *компонента Library* .dll, вызывает страницу гипертекста *Home* .html и *файл* помощи *Search* .hlp, а исходный текст этого исполнимого *компонента* хранится в файле *Control* .cpp. При этом характер отдельных видов зависимостей может быть отмечен дополнительно с помощью текстовых стереотипов.

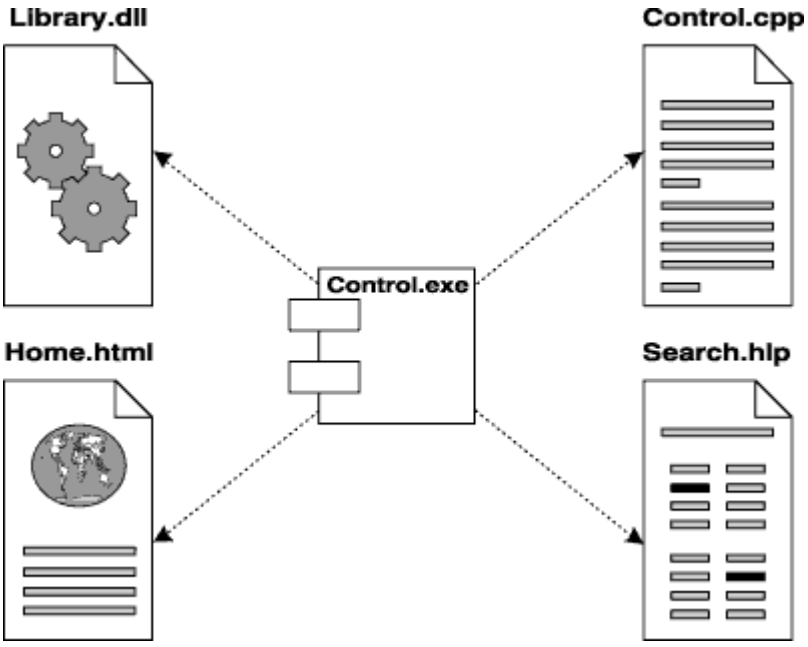


Рис. 12.5. Графическое изображение отношения зависимости между компонентами.

На диаграмме компонентов могут быть также представлены отношения зависимости между *компонентами* и реализованными в них классами. Эта *информация* имеет *значение* для обеспечения согласования логического и физического представлений модели системы. Разумеется, изменения в структуре описаний классов могут привести к изменению этой зависимости. Ниже приводится фрагмент зависимости подобного рода, когда исполнимый *компонент Control* .exe зависит от соответствующих классов (рис. 12.6).

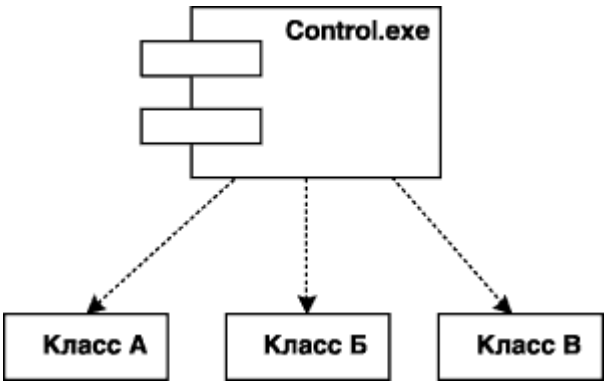


Рис. 12.6. Графическое изображение зависимости между компонентом и классами.

В этом случае из *диаграммы компонентов* не следует, что классы реализованы данным *компонентом*. Если требуется подчеркнуть, что некоторый *компонент* реализует отдельные классы, то для обозначения *компонента* используется расширенный символ прямоугольника. При этом *прямоугольник компонента* делится на две секции горизонтальной линией. Верхняя секция служит для записи имени *компонента* и, возможно, дополнительной информации, а нижняя секция – для указания реализуемых данным *компонентом* классов (рис. 12.7).

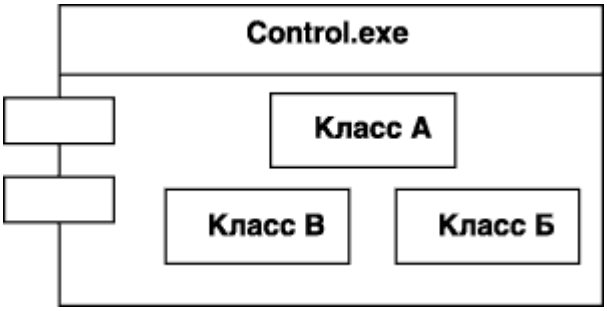


Рис. 12.7. Графическое изображение компонента с информацией о реализуемых им классах.

В случае если *компонент* является экземпляром и реализует три отдельных объекта, он изображается в форме *компонента* уровня экземпляров (рис. 12.8). Объекты, которые находятся в отдельном компоненте-экземпляре, изображаются вложенными в символ данного *компонента*. Подобная вложенность означает, что выполнение *компонента* влечет за собой выполнение операций соответствующих объектов. При этом существование *компонента* в течение времени исполнения программы обеспечивает функциональность всех вложенных в него объектов. Что касается доступа к этим объектам, то он может быть дополнительно специфицирован с помощью видимости, подобно видимости пакетов.

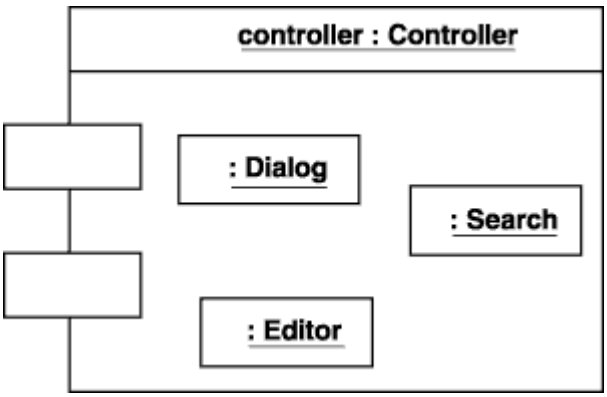


Рис. 12.8. Графическое изображение компонента-экземпляра, реализующего отдельные объекты.

Для *компонентов* с исходным текстом программы видимость может означать возможность внесения изменений в соответствующие тексты программ с их последующей перекомпиляцией. Для *компонентов* с исполняемым кодом программы видимость может характеризовать возможность запуска на *исполнение* соответствующего *компонента* или вызова реализованных в нем операций или методов.

Рекомендации по построению диаграммы компонентов

Разработка *диаграммы компонентов* предполагает использование информации не только о логическом представлении модели системы, но и об особенностях ее физической реализации. В первую *очередь*, необходимо решить, из каких физических частей или файлов будет состоять программная система. На этом этапе следует обратить внимание на такую реализацию системы, которая обеспечивала бы возможность повторного использования кода за счет рациональной декомпозиции *компонентов*, а также создание объектов только при их необходимости.

Общая *производительность* программной системы существенно зависит от рационального использования вычислительных ресурсов. Для этой цели необходимо большую часть описаний классов, их операций и методов вынести в динамические библиотеки, оставив в исполняемых *компонентах* только самые необходимые для инициализации программы фрагменты программного кода.

После общей структуризации физического представления системы необходимо дополнить модель интерфейсами и схемами *базы данных*. При разработке *интерфейсов* следует обращать внимание на согласование различных частей программной системы. Включение в модель схемы *базы данных* предполагает спецификацию отдельных таблиц и установление информационных связей между ними.

Завершающий этап построения *диаграммы компонентов* связан с установлением и нанесением на диаграмму взаимосвязей между *компонентами*, а также отношений реализации. Эти отношения должны иллюстрировать все важнейшие аспекты физической реализации системы,

начиная с особенностей компиляции исходных текстов программ и заканчивая исполнением отдельных частей программы на этапе ее выполнения. Для этой цели можно использовать различные графические стереотипы *компонентов*.

При разработке диаграммы *компонентов* следует придерживаться общих принципов создания моделей на языке *UML*. В частности, в первую *очередь* необходимо использовать уже имеющиеся в языке *UML* и общепринятые графические и текстовые стереотипы. В большинстве типовых проектов этого набора достаточно для представления *компонентов* и зависимостей между ними.

Если же проект содержит физические элементы, описание которых отсутствует в языке *UML*, то следует воспользоваться механизмом расширения. В частности, можно применить дополнительные стереотипы для отдельных нетиповых *компонентов* или *помеченные значения* для уточнения отдельных характеристик *компонентов*.

Наконец, следует обратить внимание на то, что *диаграмма компонентов*, как правило, разрабатывается совместно с диаграммой развертывания, на которой представляется *информация* о физическом размещении *компонентов* программной системы по ее отдельным узлам.

Практическое задание. Используя предметные области (см. список ниже), выполните следующие действия:

1. Определите, какие физические объекты вы создадите в проекте – файлы БД, модули, библиотеки и прочее.
2. В графическом редакторе (удобном лично Вам) создайте с использованием стандартов языка UML эти объекты на диаграмме компонентов.

Определите связи между компонентами и нанесите их на Вашу диаграмму.

Лабораторная работа №13. Осуществление взаимосвязи между диаграммами.

Теоретический материал.

UML обеспечивает поддержку всех *этапов жизненного цикла* ИС и предоставляет для этих целей ряд графических средств – диаграмм.

На этапе создания *концептуальной модели* для описания бизнес-деятельности используются *модели бизнес-прецедентов* и диаграммы видов деятельности, для описания *бизнес-объектов – модели бизнес-объектов* и *диаграммы последовательностей*.

На этапе создания логической модели ИС описание требований к системе задается в виде модели и описания системных *прецедентов*, а предварительное проектирование

осуществляется с использованием *диаграмм классов*, диаграмм последовательностей и диаграмм состояний.

На этапе создания *физической модели* детальное проектирование выполняется с использованием *диаграмм классов*, *диаграмм компонентов*, *диаграмм развертывания*.

Ниже приводятся определения и описывается назначение перечисленных диаграмм и моделей применительно к задачам *проектирования ИС* (в скобках приведены *альтернативные* названия диаграмм, использующиеся в современной литературе).

Диаграммы прецедентов (*диаграммы вариантов использования*, *use case* diagrams) – это обобщенная модель функционирования системы в окружающей среде.

Диаграммы видов деятельности (диаграммы деятельностей, *activity diagrams*) – модель бизнес-процесса или поведения системы в рамках *прецедента*.

Диаграммы взаимодействия (*interaction* diagrams) – модель процесса обмена сообщениями между объектами, представляется в виде диаграмм последовательностей (*sequence diagrams*) или кооперативных диаграмм (*collaboration* diagrams).

Диаграммы состояний (statechart diagrams) – модель динамического поведения системы и ее компонентов при переходе из одного состояния в другое.

Диаграммы классов (*class diagrams*) – *логическая модель* базовой структуры системы, отражает статическую структуру системы и связи между ее элементами.

Диаграммы *базы данных* (*database* diagrams) — модель *структуры базы данных*, отображает таблицы, столбцы, ограничения и т.п.

Диаграммы компонентов (*component* diagrams) – модель иерархии подсистем, отражает физическое *размещение* баз данных, приложений и интерфейсов ИС.

Диаграммы развертывания (диаграммы размещения, *deployment* diagrams) – модель физической архитектуры системы, отображает аппаратную конфигурацию ИС.

На рис. 1 показаны отношения между различными видами диаграмм *UML*. Указатели стрелок можно интерпретировать как *отношение* "является источником входных данных для..." (например, *диаграмма прецедентов* является источником данных для диаграмм видов деятельности и последовательности). Приведенная схема является наглядной иллюстрацией итеративного характера разработки моделей с использованием *UML*.

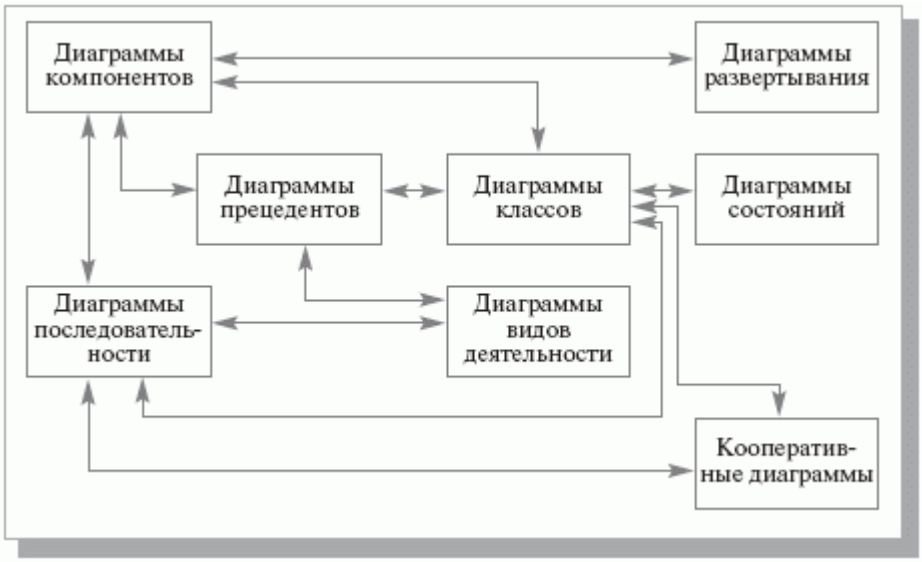


Рис.1. Взаимосвязи между диаграммами UML

Практическое задание. Используя предметные области (см. список ниже), выполните следующие действия:

1. Проверьте правильность ранее построенных диаграмм классов, взаимодействия, компонентов.
2. Убедитесь, что между диаграммами взаимодействия и классов связь правильная – должно присутствовать соответствие объектов. Если его нет, приведите одну из диаграмм в соответствующий вид.

Проверьте, соответствуют ли эти две диаграммы диаграмме компонентов и соответственно поправьте её, если есть несоответствия.

Лабораторная работа №16. «Создание каскадных таблиц стилей»

Теоретический материал.

Что такое стиль?

Для придания HTML-документу определенного внешнего вида, т.е. его оформления, каждый элемент HTML-документа имеет атрибут под названием "стиль" (style). Стиль - это правило,описывающее форматирование отдельного элемента на странице. Свойства атрибута "стиль" позволяют задать тип, размер и цвет шрифта, расположение элементов HTML-документа на экране монитора, сделать элемент невидимым или снова вернуть ему видимость

и многое другое.

Что такое CSS?

Для описания внешнего вида веб-страницы используется язык CSS. Аббревиатура CSS расшифровывается следующим образом: Каскадные Таблицы Стилей(Cascading Style Sheets).

CSS — формальный язык описания внешнего вида документа, который используется как средство описания, оформления внешнего вида веб-страниц.

CSS и HTML - это два разных языка для разных целей. Язык HTML-кода отражает смысловое содержание и логическую структуру веб-страницы. Язык CSS используется для описания внешнего вида, дизайна веб-страницы.

Например, чтобы задать размер шрифта и его цвет для одного конкретного абзаца текста в рамках документа, нужно записать код:

В этом примере внутри тэгов style расположен стиль, задающий красный цвет текста и его размер (25px).

Или: установить элементу с идентификатором id="text" красный цветтекста с помощью атрибута style. Пишем:

<script>

document.getElementById("text").style.color = "red";//св-во color объекта style

</script>

Текст, выведенный с помощью идентификатора text, будет красным.

что можно сделать с помощью стиля

Как было указано выше, стиль позволяют задать тип, размер и цвет шрифта, расположение элементов HTML-документа на экране монитора, сделать элемент невидимым или снова вернуть ему видимость и многое другое.

задание стиля внутри кода элемента

(встроенный стиль)

Проще всего записать стилевое описание непосредственно в самом элементе. Оно включает перечисление наименований свойств атрибута "стиль" и их значений внутри кода элемента. Таблица стилей, заданная внутри элемента HTML при помощи атрибута style, называется встроенной. Если вы решите изменить цвет и размер шрифта абзаца каком-то месте документа, то можно записать

далее идет текст

Если точно такое же изменение потребуется в другом месте документа, придется повторить указанный код.

Для разделения элементов списка между собой используется знак "точка с запятой" (;). Например, для задания расположения элемента в окне для отступа сверху мы пропишем следующее: margin-top: 10px. Для отступа справа: margin-right:

0px. Для задания отступа снизу: margin-bottom: 0px.

Для задания отступа слева:

margin-left: 40px.

Можно все объединить, прописав следующее: margin: 10px 0px 0px 40px;

Цифры после margin означают отступы: сверху, справа, снизу и слева.

Например, для картинки код стиля будет выглядеть так.

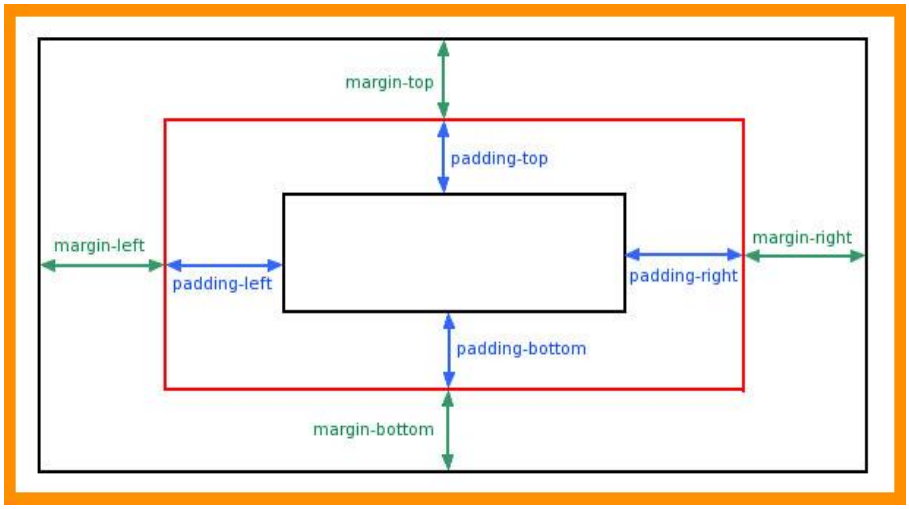
<img SRC='18.jpg' style='width:375; height:250; margin-top:10px;

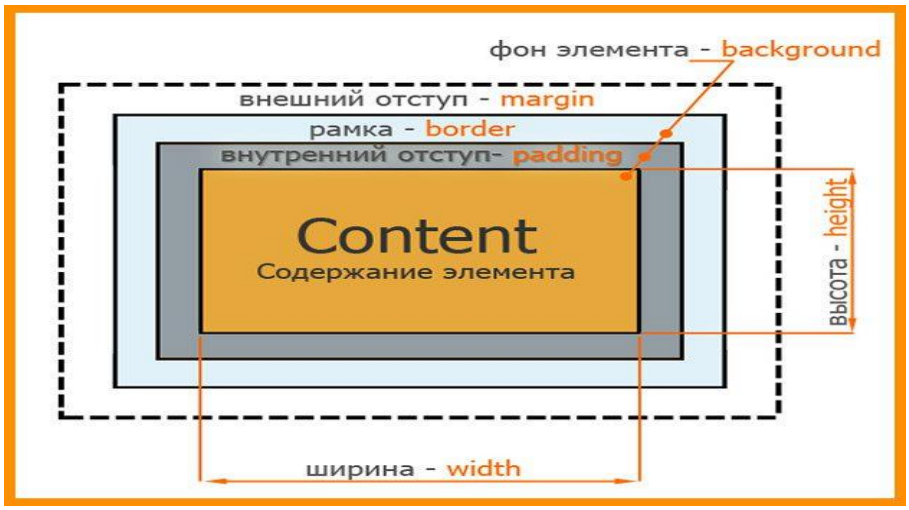
margin-left:50px;'>

Можно сократить код стиля:

<img SRC='18.jpg' style='width:375; height:250; margin: 10px 0px 0px 50px;'>

Эти коды помещают в body между и .При наличии в окне браузера большого количества элементов величины margin-top могут принимать отрицательные значения.





задание стиля вне кода элемента

(внутренний стиль)

Второй способ задания стиля именуется «внутренний стилем». При использовании встроенного стиля свойства элемента достаточно указать только один раз между тегами <style> и </style> после кода </head>. Задав набор параметров (стилей) для каждого отдельного элемента сайта, в дальнейшем нужно будет просто указать его наименование, а не прописывать все характеристики каждый раз полностью.

Набор параметров (стилей) образует список, элементы которого можно записывать как в одну строку, так и в несколько. Начинается список строкой, содержащей имя элемента и открывающую фигурную скобку (закрывающая скобка указывает на конец списка). Для разделения элементов списка между собой используется знак "точка с запятой" (;). Будьте внимательны: забыли одну скобку или двоеточие — все: ничего работать не будет, стиля как будто не существует.

Как указать элемент, к которому применяется встроенный стиль? Это можно сделать двумя способами. Либо задать идентификатор внутри элемента HTML при помощи атрибута id, либо задать класс с помощью атрибута class внутри элемента HTML.

Если стиль задается через идентификатор, то в первая строка списка будет содержать знак диеза # и имя самого идентификатора. Если вместо идентификатора используется класс, то строка начинается с точки . и к ней приписывается имя класса. Применительно к стандартным элементам (table, img, tr и т.п.) точку не ставят.

Примеры использования "id" и "class" применительно к стилю рассмотрены в дальнейших примерах.

создаем документ средствами CSS

Проиллюстрируем задание стиля следующим примером. Создадим несколько блоков, которые отобразятся в окне браузера как текст и изображение.

Последнему создадим некоторое обрамление в виде рамок и фона (две рамки, пространство между которыми (фон) сделаем цветным). Как мы помним встроенный стиль, который мы будем а основном использовать открывается тегом <style>. Далее создаем внешнюю и внутреннюю рамку.

```
<style>
#kvad1
{
   border: 6px solid red;
   width: 600px;
   height: 400px;
margin-left: 140px;
```

```
margin-top: 80px;
background-color:cyan;
}
#kvad2
{
   border: 6px solid yellow;
   width: 500px;
   height: 300px;
margin-left: 33px;
margin-top: -3px;
background-color:#c71585;}
```

Как видим, внешняя рамка поименована как kvad1, внутренняя как kvad2. Мы получили два блока, обращение к которым идет через идентификатор (знак#). Для каждой рамки мы указали ее размер, толщину, цвет, отступы слева и сверху от краев окна браузера, цвет фона между рамками. Следует учесть, что значения отступов имеют условный характер.

Чтобы эти блоки показались на экране сразу начнем формирование соответствующих тегов в body.

```
<body>
<div id="kvad1">
<div id="kvad2"><img id="card" src="18.jpg">
</div></div>
```

Имя блока помещается между тегами и . Открыв два открывающих тега <div, мы сразу же не забываем записать два закрывающих тега . Поскольку картинка будет помещена во внутреннюю рамку, т.е. в блок #kvad2, ее код должен следовать сразу за кодом этого блока. Стиль картинки (ее размеры, позиционирование) зададим в встроенном стиле (выше body). Присвоим ее блоку наименование #card.

Вообще-то img - стандартный элемент и можно было бы использовать img как наименование блока (без знака #). Но если у вас на странице есть еще картинки, они мгновенно попадут под этот стиль (например, станут одного размера). Поэтому мы и выбираем наименование #card.

```
#card
{
```

```
width: 320px;
   height: 200px;
margin-top: 25px;
}
```

Перейдем теперь к текстовым блокам. Два из них зададим как идентификаторы #title1 и #title2. Их свойства опишем в внутреннем стиле (выше body). Для третьего(выше body). Для третьего используем встроенный стиль (внутри его кода, записанного в body.

```
#title1
{
border: 6px solid #5000aa;
       border-radius: 10px;
 width: 740px;
   height: 60px;
margin-top:-40px;
text-align: center; font-size:45px;color: yellow;
font-weight: bold;
}
#title2
{
margin-top:-30px;
text-align: center; font-size:45px;color: red;
font-weight: bold;
}
</style>
```

Поскольку блок #title2 завершает внутренний стиль, после него не забываем поставить закрывающий тег </style>.

Третий текстовый элемент поместим между тегами <p и </p> в body. Кодировка body примет следующий окончательный вид:

```
<body>
<p id="title1">Пример использования стилей</p>
<p id="title2">Эта строка имеет внутренний стиль</p>
<div id="kvad1">
```

```
<div id="kvad2"><img  src="18.jpg">
<p style='margin-top:-530px;font-family: Arial;
text-align: center; font-size:40px;color: blue;font-weight: bold;'<Эта строка
имеет встроенный стиль</p>
</div></div>
</body>
```

создание с помощью стиля изображений

Средствами чистого CSS можно создавать изображения. Приведем некоторые примеры. Все отступы работают только для этой страницы.

круг

```
</head>
<body>
<div id="ita" style="  border-radius: 70px; width: 70px; height: 70px;
background-color: #0000ff; margin-left:20; margin-top:0">
</div>
</body>
```

круг с окантовкой

```
</head>
<style>
.round {
margin-left: 30;
        width: 70px;
      height: 70px;
border-radius: 70px;   /*параметры круга*/
      background: cyan;   /*заливка*/
      border: 12px solid red;   /*окантовка*/
```

```
}
</style>
<body>
<div class="round"></div>

        треугольник

</head>
  <style>
  #itak {
  width: 0px;
  height: 0px;
    border-top: 20px solid transparent;
  border-left: 45px solid transparent; //растягивает по горизонтали влево
  border-right: 45px solid  transparent; //растягивает по горизонтали вправо
  border-bottom: 75px solid  red; //растягивает по вертикали
}
</style>
<body>
<div id="itak">
</div>
</body>

перевернутый треугольник

</head>
<style>
#kati {
  width: 0;
  height: 0;
```

```
        border-left: 40px solid transparent;
        border-right: 50px solid transparent;
        border-top: 90px solid red;
margin-left:30;
margin-top:0;
}
</style>
<body>
<div id="kati"></div>
```

свой дизайн кнопки обработчика

С помощью стиля можно придать разнообразный дизайн кнопкам обработчиков. Самый примитивный способ придать кнопке неординарный вид - задать в ее коде стиль фона, бордюра, цвета и размера шрифта. С помощью размера шрифта можно регулировать размеры кнопки - ее длину и высоту. Все это делается с помощью простого кода.
<body>

```
<input  type="button" style='font-size:30px; color:purple;
border:6px solid red;  background-color:cyan '
value=пуск  onClick="alert('Кнопка работает')"></body>
```

Более изощренный дизайн требует стиля с большим количеством элементов.

Можно создать кнопку, используя только встроенный стиль (чистый CSS). Например, следующий код создаст такую кнопку

```
<style>
#itak112
{border-radius: 10px;
border:6px solid red;
background-color: cyan;
text-align:center;
```

```
padding:1px;
 width: 85px;
height: 57px;
 margin-top:-100px;
 margin-left:900px;
 color:black;
-webkit-box-shadow: 0px 6px 0px #ff0000;
-moz-box-shadow: 0px 6px 0px #ff0000;
box-shadow: 0px 6px 0px #ff0000;
}
</style>
```

При этом в body следует записать

```
<body>
<div id="itak112"><input  type=button  style=' font-size:40px;
background-color: transparent;'
value=OK   onClick="Show()">
 </div>
```

Можно создать кнопку методом document.write в скрипте. В следующем примере так создается кнопка "ВОЗВРАТ". Она появляется в новом окне после нажатия кнопки "ПУСК", которая обычным способом записана в body .

Кнопка "ВОЗВРАТ" является сочетание кода ячейки в таблице и кода изображения.

Кроме того, она выполняет функцию ссылки (сама кнопка расположена чуть ниже).

```
<script>
function WWA()
{
document.write('<table  style="background-color: cyan; border-radius: 20px; "
border=8> ');     //строку не разрывать
document.write('<tr><td ><a href="jav21.html"><img src="9.jpg"></a><font size=5
color=red><BR>ВОЗВРАТ</td></tr>')     //строку не разрывать
document.write('<BR><BR></table>');
}
```

```
</script>

</head>

<body>

<input type="button" value="ПУСК" style='border:8px  solid  red;

background-color: yellow; border-radius: 20px; font-size:30px; color:green'

onclick=WWA()>
```

Наиболее эффектные кнопки обработчиков включают в себя изображение (картинка). Это касается и последней кнопки "ВОЗВРАТ", и кнопки ОК в заголовке этого раздела. Главные коды для кнопки в заголовке приводятся ниже.

```
<style>

#itaka111

{

width: 65px;

margin-top:-120px;

 margin-left:25px;

 }

</style>

<script>

qw="Кнопка работает"

</script>

</body>

<div id="itaka111" onClick=alert(qw)><img src='111.png'>
```

У кнопки ОК в заголовке простой встроенный стиль, задающий ширину и местоположение на экране браузера. Код изображения (img) записывается в body.

Практическое задание.

Создать нижеследующий документ HTML.

```
<!DOCTYPE HTML>

 <html>

 <head>

 <meta charset="utf-8">
```

```
 </head>

 <body>

<H1> Маркированный список</H1>

<UL>

<LI>Первый пункт списка</LI>

<LI>Второй пункт списка</LI>

<LI>Третий пункт списка</LI>

<LI>Четвертый пункт списка</LI>

<LI>Пятый  пункт списка</LI>

</UL>

<H1> Нумерованный список</H1>

<OL>

<LI>Первый пункт списка</LI>

<LI>Второй пункт списка</LI>

<LI>Третий пункт списка</LI>

<LI>Четвертый пункт списка</LI>

<LI>Пятый  пункт списка</LI>

</OL>

<H1> Параграф</H1>

<P>Это содержание параграфа. Это содержание параграфа. Это содержание параграфа. Это содержание параграфа. Это содержание параграфа. Это содержание параграфа. Это содержание параграфа. Это содержание параграфа. Это содержание параграфа. Это содержание параграфа. Это содержание параграфа. </P>

  </body>

</html>
```

В заголовок документа (HEAD) вставить таблицу стилей. Посмотреть в браузере результат.

```
<STYLE type="text/css">

h1  {background-color:rgb(140,220,200)}  ul
{background-color:rgb(200,160,200)}        ol
{background-color:#DAFFC3}

p {background-color:grey}

</STYLE>
```

Сохранить документ под именем less1css.html. Открыть графический редактор и нарисовать примерно следующий рисунок размером около 100x100 пикселей. Рисунок должен быть сделан в очень бледных тонах. Сохранить его под именем smale.png в том же каталоге, что и документ.

Задать в таблице стилей фоновый рисунок smale.png для тела документа.

```
body {

}
background-image:url(smale.png);
```

Добавить к селектору body еще одно свойство.

```
background-repeat:no-repeat;
```

Изменить значение no-repeat на repeat-x. Затем на repeat-y. В чем разница? Вернуть свойство в значение no-repeat.

Добавить к селектору body еще одно свойство. Меняйте его значение и наблюдайте результат.

```
background-position:bottom center;
```

Возможные значения background-position:

top left
top center
top right
center left
center center
center right
bottom left

bottom center
bottom right
x-% y-%
x-pos y-pos

Задание 2. Параметры CSS для текста

Привести содержание документа less1css.html в первоначальный вид (как в начале урока).

В заголовок документа (HEAD) вставить «таблицу стилей».

```
<style
 type="text/css">      p
 {color: green}
 ul {color: #dda0dd}
 ol {color: rgb(0,0,255)}
</style>
```

Добавить установку цвета для заголовков документа.

Задать фоновый цвет для части текста в параграфе.

Код в таблице стилей:	span.back { background-color: gray }
Контейнер в теле документа:	....

Изменить интервал между символами.

```
p      {letter-spacing:
        1cm}          li
        {letter-spacing
        : 5px}
```

Выровнять нумерованный список по центру.

```
ol {text-align: center}
```

Задать для списков декорации.

ul {text-decoration: overline}

ol {text-decoration: line-through}

В тело документа (BODY) добавить нижеследующий код.

```
<pre style="text-transform: uppercase;">Верхний регистр</pre>
<p style="text-transform: lowercase;">Нижний регистр</p>
<pre style="text-transform: capitalize;">первые буквы в словах заглавные</pre>
```

Работа с абзацами

1. Создать файл 3.html:

```
<!DOCTYPE HTML>
<html>
    <head>
        <meta charset="windows-1251" />
    </head>
    <body>
        <p>
            По умолчанию браузер визуально отделяет абзацы друг от друа вертикальным отступом, что эквивалентно установке свойств margin-top и margin-bottom в некоторое отличное от нуля значение.
        </p>
        <p>
            В данном примере этим свойствам задано небольшое значение (0.2 em, т.е. 20% высоты символа M).
        </p>
        <p>
            Кроме того каждый абзац имеет отступ первой строки, который задан CSS-свойством  text-indent.
        </p>
```

Создать файл less3.css:

p { textindent: 2.5em; margintop: 0.2em; marginbottom: 0.2em; }

Подключить внешнюю таблицу стилей к html-документу less3.html, прописав в контейнере head следующее:

```
<link rel="stylesheet" href="less3.css" />
```

Добавить в html-документ еще пару абзацев:

```
<p class="letter">
    Буквица - еще один эффект, применяемый для визуального выделения начала абзаца. В CSS он может быть достигнут путем применения специфического стиля к первой букве абзаца.
</p>
<p class="letter">
    Первой букве абзаца соответствует селектор псевдоэлемента p:first-letter. Для этого псевдоэлемента в данном примере назначена высота 2em и красный полужирный шрифт.
```

В таблицу стилей добавить описание для псевдоэлемента класса letter:

2. strong, em, span

```
<p>
    Элемент strong по умолчанию делает вложенный текст полужирным, em -
<em>курсивом</em>, но при помощи стилей такое поведение <strong>легко переопределить</strong>. Элемент span по умолчанию никак не влияет на вложенный в него текст - к нему всегда нужно <span class="accent">применять стиль</span>.
```

7.

```
strong {
    color: red;
}

span.accent {
    background:
```

8. Логическое форматирование

```
<p><abbr title="World Wide Web">WWW</abbr></p>
```

Чтобы увидеть расшифровку аббревиатуры, надо навести мышь.

9. Моноширинный текст

```
<p>Так выглядит <tt>моноширинный текст</tt>.</p>
```

Лабораторная работа №17. Подключение каскадных таблиц к веб-странице

Теоретический материал.

Как заменить одну картинку другой

Замену одной картинки на другую можно произвести либо с помощью чистого CSS, либо с помощью чистого HTML. Рассмотрим два самых простых способа.

Чистый HTML

```
<body>

<p><img src="531.jpg" width="150" height="100"

onmouseover="this.src='18.jpg'; this.width=450;this.height=300;"

onmouseout="this.src='531.jpg'; this.width=150;this.height=100;" </p>

</body>
```

Кодировка this.src означает, что при наведении курсора (onmouseover) на "это изображение", т.е. на то, которое в коде записано перед этим (531.jpg), оно заменяется на другое (18.jpg). Одновременно размеры "этого изображения" меняются на другие. Аналогично при уводе курсора (onmouseout) происходит возврат к исходному изображению и изменение размера картинки. Возможность при смене картинки менять ее размеры - большое преимущество этого метода.

Чистый CSS

```
<style>

<!--задаем 1-ый(tim) и 2-ой(tam) рисунки. tam пустой, но будет записан в body>

.tam { }

img {margin-top:85px;              //задает расположение изображения

width: 300px;                      //размер изображения

height: 200px;

}

.tim {display:block; position:absolute;}     //полная видимость

.tim:hover {opacity:0}           //полная прозрачность-рис. исчезает

</style>

<body>

<div

<img class="tim"  src="150.jpg">

  <img class="tam" src="03.jpg">

</div

</body>
```

В этом примере смена картинки происходит только за счет встроенного стиля, т.е. чистого CSS. При наведении курсора (этому соответствует кодировка .tim:hover, которая будет разъяснена ниже в разделе "Как применить hover сразу для нескольких div-блоков") происходит изменение прозрачности: 1-ая (верхняя) картинка становится прозрачной, т.е.

невидимой {opacity:0}. 2-ая (нижняя) картинка становится видимой. При уводе курсора приходим к исходному состоянию.

Размер картинок можно задавать произвольно в блоке img, но он будет одинаков для обеих картинок. В принципе, каждой картинке можно задать свои размеры в div-е, но тогда однакартинка будет вылезать из-под другой.

Чистый JavaScript

Для полноты картины рассмотрим смену картинок с помощью чистого JavaScript (для сокращения кода в body размеры картинки перенесены во встроенный стиль.

```
<head>
<style>
.img {
width: 300px;
height: 200px;
}
</style>
<script>
function mouseOver(){
document.getElementById("button").src="03.jpg";}
function mouseOut(){
document.getElementById("button").src="18.jpg";}
</script>
</head>
<body>
<img class="img" src="18.jpg" id="mouse" onmouseover="mouseOver()"
onmouseout="mouseOut()"></a>
```

В этом примере задаются две функции (MouseOver и MouseOut), в каждой из которых записан код вывода изображения с помощью метода getElementBy(Id), используя идентификатор mouse.

Используя события onmouseover и onmouseout, при наведении курсора выводится одна картинка, при уводе курсора - другая.

Смена фоновой картинки

```
<style>
.trim{
margin-top: -280px;        //значения условные
margin-left: -180px;       //значения условные
display: block;
     background:url("18.jpg") center ;  //первый рисунок
background-repeat: no-repeat ;
width: 600px;
  height: 400px;
opacity: 1;
                }
.trim:hover{

   background:url("531.jpg") center;
background-repeat: no-repeat;
background-size: 100% 100%;                    //второй рисунок
}
</style>
<body>
<div id="trim"   ></div>
</body>
```

Особенность фоновых картинок состоит в том, что они создаются исключительно средствами CSS. Поэтому вызывают изображение по единому электронному адресу ресурсов в интернете (url), который указывают внутри блока встроенного стиля. Для задания размеров картинки надо либо указать либо ширину (width) и высоту (height) в пикселях (px), либо соответствующие им проценты в коде background-size. Кодировка background-repeat: no-repeat устраняет повторение картинки на экране браузера в случае малого размера картинки. Замене картинок путем наведения курсора соответствует блок #tim:hover.

Дополнительно приведем кодировку смены картинок, которая сопровождается увеличением размера второй картинки. Кроме того, размер первой картинки уменьшен по сравнению с оригиналом (30% от width=600px и height=400px).

```
<style>
#tim
```

```
{
width:600px;
height:400px;
 margin-left:20px;
 margin-top:10px;
background-image:url("77.jpg");
background-repeat: no-repeat;
background-size: 30% 30%;
}
#tim:hover
{
background-image:url("60.jpg");
background-repeat: no-repeat;
background-size: 100% 100%;
}
</style>
<body>
<div id="tim"   ></div>
</body>
```

Анимационные эффекты с помощью CSS

С помощью CSS можно проделывать не просто разнообразные изменения элементов. Этим изменениям можно придать эффект растянутости во времени, т.е. наделить их свойством двигаться. Осуществляется это с помощью "движков", которым соответствуют определенные коды. Если у вас не очень старая версия браузера, то вы используете код -webkit-transition, -webkit-transform и другие, начинающиеся со слова -webkit. Для поддержки ранних версий браузеров использовались коды движков, начинающихся с -moz или -ms. Многие авторы в интернете в документе вываливают сразу коды всех движков. Вам же нужно убедиться, какой именно один из трех переваривает ваш браузер и ограничиться одним кодом.

Растягивание элемента

В этом примере рассматривается изменение размеров квадрата, но попутно добавлено изменение цвета рамки и фона. Если бы речь шла только об изменении размера, нужно использовать кодировку -webkit-transition: width 2s, height 2s;

Это означает, что для изменения ширины и высоты квадрата мы задали время 2 секунды. Затем в кодировку #wox1:hover{} внутри фигурных скобок мы прописываем, какие именно значения ширины и высоты мы хотим иметь после окончания движения:

#wox1:hover {width: 250px; height: 250px;}

Если одновременно с размерами, мы хотим изменить вдобавок еще и другие свойства, мы вписываем их в фигурные скобки. При этом в коде "движка" -webkit-transition мы должны были бы перечислить эти свойства. Но можно не заморачиваться этим, а просто записать -webkit-transition: all 2s;

```
<style>
#wox1 {
    display: block;
    width: 100px;
    height: 100px;
margin-left: 10px;
margin-top: -80px;
      background-color: pink;
 border: 6px solid red;
   -webkit-transition: all 2s;
}

#wox1:hover {
   background-color: #00FF00;
    width: 250px;
    height: 250px;
border: 6px solid yellow;

}
</style>
<script>
function FF()
{
```

```
document.getElementById('wox1').style.display ='block'

}

</script>

<body>
```

Затухание (медленное исчезновение) элемента

Свойство transition (в переводе на русский означает «переход») придает изменению стиля растянутость во времени (плавность перехода). К примеру, цвет блока при наведении на него мышкой изменится не резко, а в течение определенного времени.

Элемент как бы растворяется в воздухе. Код эффекта очень простой:-webkit-transition:5s; он указывает, что переход от полной видимости (непрозрачности) до полной невидимости (полной прозрачности) происходит за 5 секунд. Начинается переход при наведении курсора в соответствии с кодом

#tram:hover{opacity:0}.

```
<style>

#tram{

 width:450px;

 height:300px;

-webkit-transition:5s ;      //время эффекта

}

#tram:hover{

opacity:0;              //прозрачность

}

</style>

<body>

<div id="tram"><img  src="1002.jpg"></div>
```

Поворот элемента

Для поворота элемента на 180 градусов применяем код:

-webkit-transform: rotate(180deg);

```
<style>
.box3 {

    display: block;
    width: 100px;
    height: 200px;
    background-color: #0000FF;
margin-left: 100px;
margin-top: 100px;
font-size:20px;
 color:red;
    -webkit-transition: all 2s
}
.box3:hover {
     -webkit-transform: rotate(180deg);

}
</style>
<body>
<div class="box3"><img  src="9.jpg"></div>
```

Смена картинок с затуханием

Замена одной картинки другой за счет плавного затухания (исчезновения) верхней картинки. Код эффекта очень простой (он уже был приведен выше): -webkit-transition:3s; он указывает, что переход от полной видимости (непрозрачности) до полной невидимости (полной прозрачности) происходит за 3 секунды. Начинается переход при наведении курсора в соответствии с кодом

```
 .tremf:hover{opacity:0}.

<style>
.tremf {display:block; position:absolute;}
.tremf:hover{opacity:0;
```

```
-webkit-transition:3s ;      //время эффекта
}
</style>
<body>
<div><img class="tremf"  src="03.jpg">
<img class="tremf"  src="18.jpg">
</div>
```

Смена картинок с плавным исчезновением

Еще один интересный эффект при смене картинок - "схлопывание верхней картинки". Уменьшаясь в размерах картинка стягивается в точку, однако процесс "схлопывания" происходит не мгновенно, а плавно. Задав определенное время в коде -webkit-transition: all 1s можно это "схлопывание" сделать либо мгновенным, либо значительно растянуть во времени.

```
<style>
.rom img {
margin-left: 400px;
margin-top: -300px;

    position:absolute;

  -webkit-transition: all 1s ;
  }
.rom img.raz
 {
height: 400px;
    width:600px;
   -webkit-transform:scale(0);
 }
 .rom:hover img.raz
{
```

```
 -moz-transform:scale(1);
 -webkit-transform:scale(1);
}

.rom:hover img.dva
{
     -moz-transform:scale(0);
 -webkit-transform:scale(0);
 }
</style>
<body>
<a class="rom" href="#">
<img class="raz" src='18.jpg' width = "300" height="200">
<img class="dva" src='03.jpg' width = "300" height="200">
</a>
```

Простое новое окно средствами CSS

В интернете есть много примеров создания нового (всплывающего или модального) окна. Однако в большинстве они очень громоздкие по коду и оставляют желать лучшего в отношении пояснений. Мы создадим несколько окошек, начиная с самого простого. Его кодировка такова.

```
<style>
    #new-wind {
       width: 600px;
       height: 500px;
       text-align: center;
       padding: 15px;
       border: 6px solid red;
       border-radius: 10px;
       background-color:#0f9;
position: fixed;
```

```
   top: 0px;
  right: 0;
  bottom: 0;
  left: 0px;
  margin: auto;
display: none;
      }
   #new-wind:target {display: block; }
a{text-decoration: none; font-size:20px;border: 4px solid red;}
  </style>
  </head>
   <body>
     <div id="new-wind">
<a href="#price"Закрыть окно</a>
<p3Здесь можно расположить content</p>
    </div>
 <a href="#new-wind"<Открыть новое окно</a>
</body>
```

Основному блоку присвоен идентификатор #new-wind. Напомним, что блок - это совокупность стилевых свойств, ограниченная фигурными скобками и имеющая наименование. В этом блоке мы перечисляем: широту и высоту окна, центровку текста, рамку и ее закругление, цвет фона. Далее задаем позиционирование. Наконец, поскольку в неактивном состоянии окно должно быть скрыто, последней строкой в блоке стиля окна указываем display: none. Для вывода нашего модального окна из неактивного состояния используем псевдо-

класс target. Последний изменяет режим вывода элемента, поэтому наше модальное окно будет выводиться при нажатии на ссылку. Псевдо-класс :target позволяет связать гиперссылку <a> с любым элементом на странице, при этом значение атрибута href ссылки должно начинаться с символа # и содержать либо идентификатор id выбранного элемента, либо метку (якорь). При нажатии на ссылку с псевдо-класс :target в соответствии с идентификатором или якорем перенаправляет документ к этому элементу, т. е. этот элемент становится целью (отсюда и :target- цель, мишень).

В теле документа (после body) между и перечисляем идентификаторы. Закрытие окна привязываем к внутренней ссылке с якорем. Код гиперссылки на открытие окна (которая должна быть на странице основного документы) записываем после .

Наконец в представленной кодировке имеется строка

a{text-decoration: none; font-size:20px;border: 4px solid red;}

Это стиль ссылок. В нем отменено подчеркивание и задана рамка красного цвета.

Немного усложним наше окно. Создадим стили для кнопки "закрыть" и для заголовка. Теперь они будут независимыми элементами окна и при необходимости можно внести изменения в их стиль. Кроме того, запишем в окно текст. Для разнообразия стилевые блоки для этих элементов оформим как классы. Блок заголовка и блок текста кодируются так.

```
 .title1{
margin-top:0px;margin-bottom:0;
text-align: left; font-size:30px;color: yellow;
font-weight: bold;
}
.title2
{
margin-top:0px;margin-bottom:0;
text-align: left; font-size:70px;color: red;
font-weight: bold;
}
```

Блок закрытия окна

```
.close
{
margin-left:400px;
margin-top:-330px;
padding:1px 5px 0;
border:2px solid red;
font-size:20px;
right:4px;
}
```

Блок .... нужно дополнить кодами

```
<p class="title1">Заголовок модального окна</p>
```

Здесь можно
 расположить
 content

закрыть

Посмотрите, что из этого получилось.

Теперь вложим в новое окно небольшой content. Придадим ему динамический характер (здесь одним стилем на обойтись - потребуется использование JavaScript). Посмотрите, как это выглядит.

Как задать hover сразу для нескольких элементов

При работе с изображениями широко используется псевдокласс hover. Псевдоклассы - это особые свойства, которые позволяют менять стиль элемента в зависимости от действий ользователя. Классическим примером применения псевдоклассов является ссылка, которая меняет свой цвет при наведении на неё курсора.

Если мы хотим изменить стиль элемента, это можно сделать с помощью псевдокласса hover. Чтобы наведение курсора изменяло вид элемента, используется следующий синтаксис: элемент:hover { ... }

Этот код срабатывает срабатывает, когда пользователь наводит курсор на элемент, но при этом кнопка мыши не нажата.

Иногда нужно, чтобы при наведении курсора на один элемент изменения происходили и в других. При этом может возникнуть сложность в записи кода hover сразу для нескольких элементов. Начнем с простого примера.

Пусть в встроенном стиле находится блок

```
<style>

.aa {

   border: 6px solid red;

   width: 190px;

   height: 100px;

   background-color: #0000FF;

text-align: center;

font-size:35px;

color:yellow;

  }

</style>
```

Чтобы его свойства изменились при наведении курсора, в стиль нужно добавить следующую запись:

```
.aa:hover
```

{

width: 300px;

height: 130px;

background-color: green;

font-size:0px;

}

С помощью hover мы изменили размер прямоугольника, цвет фона и отменили текст.

Усложним задачу. Пусть мы имеем три блока и хотим, наведя курсор на один блок, вызвать изменения сразу в трех. К уже имеющемуся блоку aa добавим еще два: два прямоугольника, один из которых подобен блоку aa (блок bb), а другой (dd) содержит надпись "наведи курсор на верхний квадрат".

.bb {

border: 6px solid red;

width: 210px;

height: 100px;

background-color: #0000FF;

font-size:0px;

}

.dd

{

border: 6px solid red;

border-radius: 8px;

height: 50px;

width: 330px;

text-align: center;

font-size:20;

color:green;

font-weight: bold;

background-color: yellow;

}

Сразу не забудем дать этим блокам описание в body:

текст
появился

Наведи курсор
на верхний квадрат

Применим и к новым блокам псевдокласс hover:

```
.aa:hover + .bb{
font-family:Monotype Corsiva;
text-align: center;
font-size:35px;
background-color: pink;
height: 130px;
width: 320px;
text-align:center;
font-size:45px;
color:blue;
}
.aa:hover ~ .dd{display: none}
```

Обратите внимание на запись кода hover, когда мы применяем его к нескольким элементам: в первой строчке ставим знак +, а во второй - знак подобия ~. Теперь посмотрим, что у нас получится при наведении курсора на верхний квадрат.

Мы видим, что наведение курсора на верхний прямоугольник производит изменения сразу в трех блоках: верхний блок исчезает, а в двух нижних происходят изменения с фоном, размером и текстом.

Заметим, когда нужно произвести ОДИНАКОВЫЕ изменения в трех блоках, можно было бы обойтись самым кратким кодом:

```
.a:hover, .b:hover, .c:hover{}
```

Однако все блоки вели бы себя как независимые, т.е. курсор вызывал изменения только в наведенном блоке.

Как прикрепить CSS к скрипту

Рассмотрим, как задается CSS для элементов скрипта. С помощью метода document.write создадим две таблицы: одну с внутренним стилем (вне кода элементов) между тегами </head> и <script>), другую - с встроенным стилем (внутри кода элементов). Еше две таблицы, идентичные первым двум, записаны внутри функции и призваны продемонстрировать специфические взаимоотношения CSS и метода document.write. Также скрипт содержит коды, соответствующие выводу и скрытию элементов с задержкой и без нее.

</head>

```
<style>
#xox
{
display: block;
}
#xox1
{
display: none;
}

#tab2
{
border-color:red ;
border-style: solid;
border-width: 6px;
width:160px;
height:160px;
font-size:25px;
color:blue;
text-align: center;
}
#td2a
{
border-color:purple ;
border-style: solid;
border-width: 4px;
 background-color: cyan;
}
#td2b
{
```

```
border-color:purple ;
border-style: solid;
border-width: 4px;
 background-color: pink;
}
#td2c
{
border-color:purple ;
border-style: solid;
border-width: 4px;
 background-color: cfc;
}
#td2d
{
border-color:purple ;
border-style: solid;
border-width: 4px;
 background-color: green;
}

</style>
<script>
tab0="<TABLE id='xox'><TD>"
tab2="<TABLE id='tab2'>"
sj5="<td id='td2a'>ячейка 1</TD>"
sj6="<td id='td2b'>ячейка 2</TD><TR>"
sj7="<td id='td2c'>ячейка 3</TD>"
sj8="<td id='td2d'>ячейка 4</TD></TABLE>"
tab3="<TABLE width=160px height=160px   style=' border-color:red; border-style: solid;
border-width: 6px;text-align: center;'>"
```

sj9="<td style='font-size:25px; color:blue; border-color:purple;border-style: solid;

border-width: 4px;background-color: cyan;'>ячейка 1</TD>"

Ячейки sj10, sj11, sj12 имеют тот же код, что и sj9, за одним исключением: они

отличаются цветом фона и нумерацией.

sj10="<td style='.....;background-color: pink;'>ячейка 2</TD><TR>"

sj11="<td style='.....;background-color: #cfc;'>ячейка 3</TD>"

sj12="<td style='.....;background-color:green;'>ячейка4</TD></TABLE></td></table>"

Ячейки собираются вместе и вместе с кодом таблицы выводятся методом document.write.

tc2=tab0+tab2+sj5+sj6+sj7+sj8

tc3=tab3+sj9+sj10+sj11+sj12

tc=tc2+tc3

document.write(tc)

Эти же самые две таблицы запишем внутри функции, которую будем запускать кнопкой.

function WW()

{

b="<p align=center><img id='kot' SRC='image.jpg' width=300 height=200 >"

setTimeout('document.write(b)',5000)

setTimeout("document.getElementById('kot').style.display='none'",12000)

setTimeout("document.getElementById('xox1').style.display='none'",6000)

document.getElementById('xox').style.display='none'Далее идет повтор кодировки двух

таблиц, записанной перед функцией.

tab0="<TABLE id='xox1'><TD>"

tab2="<TABLE id='tab2'>"

sj5="<td id='td2'>ячейка 1</TD>"

sj6="<td id='td2'>ячейка 2</TD><TR>"

sj7="<td id='td2'>ячейка 3</TD>"

sj8="<td id='td2'>ячейка 4</TD></TABLE>"

tab3="<TABLE width=160px height=160px style=' border-color:red; border-style: solid;

border-width: 6px;text-align: center;'>"

sj9="<td style='font-size:25px; color:blue; border-color:purple;border-style: solid;

border-width: 4px; background-color: cyan;'>ячейка 1</TD>"

```
sj10="<td style='.....;background-color: pink;'>ячейка 2</TD><TR>"

sj11="<td style='.....;background-color: #cfc;'>ячейка 3</TD>"

sj12="<td style='.....;background-color:green;'>ячейка4</TD></TABLE></td></table>"

tc2=tab0+tab2+sj5+sj6+sj7+sj8

tc3=tab3+sj9+sj10+sj11+sj12

tc=tc2+tc3

document.write(tc)

t="<BR><span style='font-size:25;color:blue'>Внутренний стиль        Встроенный

стиль</span><BR><BR>"

document.write(t)

w="<BR><span style='font-size:28;color:red'>После нажатия кнопки  ПУСК у таблицы

с внутренним стилем - стиль исчез, у таблицы с встроенным

стилем -

стиль сохранился.</span>"

document.write(w)

}

</script>

<body >

<input type="button" value="ПУСК" onclick="WW()">
```

При запуске функции кнопкой внутренний стиль исчез, а встроенный стиль сохранился. Это связано со специфическими взаимоотношениями CSS и метода document.write.

Маленькие хитрости CSS

Приведем ряд кодов стилевого оформления документа, редко упоминаемых в руководствах по CCS.

Как убить шрифт.

Пусть при наведении курсора шрифт должен исчезать, тогда в стиле мы используем код #a:hover{font-size:0px;}

Здесь #a - наименование блока, который содержит описание шрифта.

Как отцентрировать текст в рамке по вертикали.

Чтобы произвести центровку текста по вертикали, нужно изменить значение параметра line-height (задается цифрой в px). Например,

line-height: 40px;

Уменьшая цифры перемещаем текст вверх, увеличивая - -вниз.

Как изменить расстояние между буквами и строками.

Расстояние между буквами или цифрами можно изменить, поменяв значение параметра в коде

letter-spacing: 0.2em;

где em - это относительная единица измерения расстояние между буквами.

Расстояние между строками можно изменить, поменяв значение параметра в коде

line-height : 50px;

Вместо пикселей (px) можно использовать пункты (pt), высоту шрифта текущего элемента (em), дюймы(in), множитель (число - как на пишущей машинке)

и так далее.

Как изменить стиль ссылки.

Вместо стандартного вида ссылки можно сделать ссылку со своим собственным стилем. Например, можно убрать подчеркивание, изменить шрифт, изменить фон. Чтобы ссылка не подчеркивалась используем код:

text-decoration: none;

Чтобы изменить шрифт:

font-family:Arial; font-size:35px; color:green;

Чтобы изменить фон:

background-color: grey;

Чтобы сделать рамку:

border:4px solid red;

Если стиль ссылки содержит рамку (border), а в ссылке есть якорь, то код якоря надо записывать так

Иначе появляются две вертикальные линии с высотой с высоту рамки.

Как изменить стиль рамки.

Чаще всего используют сплошную рамку, но ей можно придать и другой вид.

Двойная. border-top: 12px purple double; border-bottom: 12px purple double;

border-left: 12px purple double; border-right: 12px purple double;

Штриховая. border: 6px dashed ;

border-color:green ;

Пунктирная. border: 6px dotted ;

border-color:blue ;

Выпуклая. border: 11px outset ;

border-color:#f0f ;

Как отменить стиль элемента.

Предположим в блоке нужно отменить выше заданный стиль. Рассмотрим это на простом примере. Пусть при наведении курсора мы хотим иметь изменения размеров двух блоков.

Назовем эти блоки: Блок А и Блок Б. Предположим, что в блоке А мы хотим растянуть изменение размеров во времени (свойство transition). Время изменения - 2 секунды.

Зададим это изменение в начале внутреннего стиля. Тогда оно будет применяться к обоим блокам, и его не надо указывать в стиле каждого блока.

*{ -webkit-transition: width 2s, height 2s;}

Но в блоке Б нам свойство transition не нужно, а поэтому в стиле этого блока его надо отменить. Для этого используем код:

-webkit-transition:none;

Теперь при наведении курсора размеры блока А будут меняться медленно, а размеры блока Б - мгновенно посмотреть 6

Что такое z-index .

Изображение каждого блока появляется на экране в том порядке, в котором они перечислены во внутреннем стиле (между тегами <head> и </head>). При этом в пространстве очередной блок располагается поверх предыдущего, т.е ближе к пользователю. Удаление изображения блока от экрана (соответственно приближение его к пользователю) задается свойством под названием z-index. Если в стиле он отсутствует, то по умолчанию он имеет нулевое значение и расположение блоков перпендикулярно плоскости экрана соответствует вышеописанному.

Если мы хотим какой-то блок удалить от экрана (приблизить к себе) нужно в стиле этого блока указать z-index=1. Если меняется расположение только одного блока, цифра задается совершенно произвольно (лишь бы не ноль). Если меняется расположение нескольких блоков, значение z-index увеличивается по отношению к исходной цифре.

Чем ближе к себе мы хотим видеть изображение блока, тем больше должно быть значение z-index.

Как вставить текст в стиль CSS

Обычно текст вставляют в body, а его параметры задают в стиле.

Можно ли текст вставить непосредственно стиль? Да, можно, если использовать, например, псевдокласс after. Приведем код, поименовав псевдокласс как aauw:after.

Запишем в стиль предложение: 'Текст, записанный в стиле(class 'aau ', псевдокласс 'after '), при использовании шрифта Segoe Script'. Код будет такой

```
.aauw:after{position: absolute; white-space: pre;

content:

"\0aТекст, записанный в стиле\0a(class 'aau ', псевдокласс\0a 'after '), при использовании  \0a
шрифта Segoe Script" ;

font-family: Segoe Script;

font-size:50px; font-weight: bold;

color: red; line-height: 2.0;}
```

О некоторых особенностях форматирования текста, записанного в стиле, смотри чуть ниже в параграфе "Как сделать перенос строки в тексте, записанном в стиле CSS"

Как сделать перенос строки в тексте, записанном в стиле CSS

Чтобы сделать перенос в строке, записанной в стиле, нужно в элемент content вставить свой текст, поместив его в кавычки. Сам код переноса - совокупность трех символов: наклонная черта, ноль и латинская буква 'a' (\0a).

В целом это выглядит так.

```
content:

"\0a Эта строка записана с переносом . \0a Сам текст, записан в стиле\0a

при использовании шрифта \0a именуемого как Segoe Script ." ;
```

Помимо этого, естественно должны быть указаны: псевдокласс (.c:after),

позиционирование, а после записи строки - характеристики шрифта. Это выглядит так:

<style>

.c:after{position: absolute; margin-top:-200px; left: 12%; margin-right: -50%;

content:

"\0a Эта строка записана с переносом . \0a Сам текст, записан в стиле\0a при использовании шрифта \0a именуемого как Segoe Script ." ;

white-space:pre;

font-family: Segoe Script;

font-size:50px; font-weight: bold;

color: green; line-height: 2.0;}

</style>

Практическое задание.

1. Напишите код предложенных элементов в CSS, подключите к существующему из предыдущих работ файлу HTML и убедитесь в работоспособности каскадного стиля.

Списки

1. **Вложенный список**

```
<!DOCTYPE HTML>
<html>
	<head>
		<meta charset="utf-8" />
		<link rel="stylesheet" href="less4.css" />
	</head>
	<body>
		<p>Первый список:</p>
		<ol>
			<li>Пункт</li>
			<li>Еще один пункт</li>
			<li>Предпоследний  пункт</li>
```

```
ol {
	list-style-type: upper-roman;
}
	list-style-image: url(one.png);
}
```

2. Идентификаторы

```
<div id="ls">
        <ul id="list1">
                <li>Первый пункт первого списка</li>
                <li>Второй пункт первого списка</li>
                <li>Третий пункт первого списка</li>
        </ul>
        <ul id="list2">
```

```
#ls {
        width: 250px;
}
        list-style-position: inside;
}
        list-style-position: outside;
}
```

Ссылки

- **link** — состояние по умолчанию, оно определяет, как должны выглядеть ссылки в определенной части документа. По умолчанию непосещенные ссылки имеют синий цвет.
- **visited** — стиль ссылки, которая уже была посещена раньше (она может находиться в кэше браузера). По умолчанию посещенные ссылки имеют пурпурный цвет.
- **hover** — стиль ссылки, когда курсор мыши находится над ней.
- **active** — стиль ссылки, когда она активируется, т.е. когда выполняется соединение с другим сайтом; это также стиль последней активированной ссылки, когда вы снова возвращаетесь к документу в своем браузере.

(Ссылки и цели)

```
<!DOCTYPE HTML>
<html>
	<head>
		<meta charset="utf-8" />
		<link rel="stylesheet" href="less4.css" />
	</head>
	<body>
		<a id="start"> </a>
		<ul>
			<li><a href="#p2">К  секции p2</a></li>
			<li><a href="#end">В конец документа</a></li>
		</ul>
		<p id="p1">
			<strong>Это текста в секции p1.</strong> Это текста в секции p1. Это текста в секции p1. Это текста в секции p1. Это текста в секции p1. Это текста в секции p1. Это текста в секции p1. Это текста в секции p1. Это текста в секции p1. Это текста в секции  p1. Это текста в секции p1. Это текста в секции p1. Это текста в секции    p1.
		</p>
		<p id="p2">
			<strong>Это текста в секции p2.</strong> Это текста в
```

Первый вариант:

```
a:link {
	color: #33ccff;
}
a:visited {
	color: #cecece;
```

```
}
a:active {
        color: #339999;
```

Второй вариант:

```
ul {
        list-style: none;
}

li a {
        color: #fff;
        text-decoration:
        none; padding: 4px
        7px; width: 120px;
        background:
        #0076ba; display:
        block;
        border: 1px solid
```

Позиционирование и обрезка

1.

```
<!DOCTYPE HTML>
<html>
<head>
        <meta charset="utf-8" />
        <link rel="stylesheet" href="less6.css" />
    </head>
    <body>
        <div id="scroll">
            Эта секция имеет значение scroll свойства overflow. Поэтому полосы прокрутки показаны, хотя и не нужны.
        </div>
        <div id="hidden">
            Эта секция имеет значение hidden свойства overflow. Поэтому непоместившийся в нее текст остается невидимым.
        </div>
        <div id="visible">
            Эта секция имеет значение visible свойства overflow. Поэтому часть текста (если его много) может выходить за ее границы.
        </div>
```

```
        </body>
    </html
    > div {
        position:  absolute;
        border:  2px  solid
        black; padding: 3px;
}
    #scroll
    {
        overflow:  scroll;
        top: 10%;
        bottom: 10%;
        left: 0;
        right: 20%;

}
    #hidden {
        overflow:  hidden;
        top: 30%;
        left:  10%;
        width:
        150px;
        height: 97px;
    }
    #visible {
        overflow:
        visible;  top:
        100px;  right:
        50px;
        max-width:
        150px;
        max-height: 97px;
    }
```

2. clip — прямоугольное окно

```
<div id="div1"></div>
```

```
#div1, #div2 {
```

```
        width: 320px;
        height: 320px;
}
#div1 {
        background-image: url(smale.png);
}
#div2 {
```

Лабораторная работа №18. Использование основных элементов и функций языка JavaScript

Цель работы: научиться использовать различные способы доступа к свойствам и методам объектов для внесения изменений в HTML-документ.

Теоретический материал.

Все изменения в HTML-документе производятся через свойства, методы и события объектов, входящих в состав объектной модели документа. Рассмотрим сценарий (пример 1.1), в котором используются типичные способы доступа к свойствам и методам объектов.

Пример 1.1.

```
<HTML ID='DOCUM'>
<CENTER><H1 ID='zag'> ВАСЬКА</h1>
Вариант, совместимый с Mozilla<P>
<FORM NAME='f1'>
<INPUT TYPE=BUTTON name="bot" onclick=bm() value='Увеличить'>
<P>
</FORM >
<IMG SRC="VaskaM.jpg" ID='Vas' onclick=bm()>
</center>
<!--В скрипте закомментирован вывод окна со списком объектов -->
<SCRIPT>
/*var
msg="
for(i=0; i<document.all.length;i++)
msg+=i + ' ' + document.all[i].tagName +' ID=' + document.all[i].id+'\n'
alert(msg) */
flag=true
function bm()
{ if(flag)

{ document.images[0].src='VaskaB.jpg' flag=false;
  document.forms[0].bot.value="Уменьшить"
  document.forms[0].bot.style.background='red'
  document.all.bot.style.color='black';
```

```
}
else
{ document.getElementById("Vas").src='VaskaM.jpg';
  //обращение к кнопке по индексу
  document.forms[0].bot.value="Увеличить"
  document.forms[0].bot.style.background='green'
  document.all.bot.style.color='white'; flag= true
}

}
</script>
</HTML>
```

В примере 1.1 сценарий используется для замены фотографии и изменения цвета кнопки и надписи на ней. Друг друга меняют маленькая и большая фотографии кота Васьки.

Замена фотографии и изменение вида кнопки в примере происходят при щелчке мышкой по кнопке или по самой фотографии. В тегах <*INPUT*> и <*IMG*> помещён атрибут*onclick=bm()*, при помощи которого в ответ на щелчок мышкой вызывается функция *bm()*. Такой атрибут называют обработчиком событий. Имя файла с фотографией является свойством *SRC* объекта *IMG*. Замена фотографии делается в сценарии двумя равнозначными способами:

```
document.images[0].src='VaskaB.jpg'
document.getElementById("Vas").src='VaskaM.jpg'
```

В первом варианте доступ к фотографии происходит через коллекцию *images*, в которой все расположенные на странице фотографии (в примере одна фотография) пронумерованы. Нумерация начинается с нуля. Во втором варианте используется метод *getElementById()* объекта *document*. Параметром в методе *getElementById()* служит *ID*(идентификатор) объекта *IMG*. Весь оператор читается так: "свойству *src* объекта, имеющего *ID=Vas* и входящего в состав объекта *document*, присвоить значение *VaskaM.jpg* ". Сам объект *document* является дочерним по отношению к объекту *window*. Поэтому полное обращение к свойству *src* следовало бы писать так:

```
window.document.images[0].src='VaskaB.jpg'
window.document.getElementById("Vas").src='VaskaM.jpg'
```

Обычно, за исключением тех случаев, когда нужно воспользоваться свойствами или методами самого объекта *window*, это слово опускают.

Более сложная иерархия объектов наблюдается при обращении к свойствам кнопки:

```
document.forms[0].bot.value="Уменьшить"

document.forms.f1.bot.style.background='red'
```

Первый оператор служит для изменения надписи на кнопке. Доступ к кнопке происходит по её имени (*name='bot'*). Доступ к форме осуществляется через коллекцию *forms* и номер формы в коллекции.

С помощью второго оператора меняется цвет кнопки. Доступ к форме осуществляется через коллекцию *forms* и имя формы *f1*. Своеобразие доступа к свойствам, задаваемым стилем, состоит в том, что *style* выступает формально как дочерний объект. В рассматриваемом примере родительским по отношению к *style* является объект с именем *bot*, то есть, кнопка.

Для обращения к объектам удобна коллекция *all* объекта *document*. В коллекцию *all* входят все объекты HTML-документа, поэтому выбирать объект нужно по его *ID* или имени. Коллекция *all* использована для изменения цвета надписи на кнопке:

```
document.all.bot.style.color='black'
```

Все описанные способы доступа к объектам HTML-документа базируются на объектной модели документа (DOM). Наиболее трудной задачей, стоящей перед разработчиком сценариев, является разработка сайтов, которые будут одинаково выполняться на всех браузерах. Рассматриваемый пример HTML-документа подобран так, что он (почти) одинаково выполняется на браузерах Internet Explorer 6.0 и Mozilla Firefox 2.0, несмотря на то, что объектные модели в этих браузерах отличаются друг от друга. Для того, чтобы убедиться в этом уберите символы комментариев /* и */ из первой и пятой строчек скрипта и откройте страницу в обоих браузерах. Появится окно со списком всех объектов, созданных браузером для открытого документа. В созданной браузером Internet Explorer модели – 12 объектов, а у Mozilla Firefox – 10.

Рассмотрим подробнее раскомментированный фрагмент скрипта.

```
var msg=''
for(i=0; i<document.all.length;i++)
msg+=i + ' ' + document.all[i].tagName +' ID=' + document.all[i].id+'\n'
alert(msg)
```

Коллекция *all* всех объектов HTML-документа имеет свойство *length*, равное количеству всех объектов модели документа. Любой объект имеет свойство *tagName*, значение которого совпадает с именем объекта. В приведённом фрагменте скрипта в цикле формируется строковая переменная *msg*, состоящая из номеров, имён и *ID* объектов. Сочетание символов *\n* служит для перевода строки.

Размещение сценариев в HTML-документе

Сценарий в HTML-документе можно размещать тремя способами:

- в открывающем теге в качестве значения атрибута событие (см. пример 2.1);
- в контейнере, ограниченном тегами *<SCRIPT> ...</script>;*
- в отдельном файле.

Контейнеры со сценариями могут размещаться в любом месте HTML-документа. Количество сценариев в одном HTML-документе не ограничено.

Выполнение сценариев происходит при загрузке HTML-документа и при наступлении событий. Во время загрузки HTML-документа браузер последовательно просматривает встречающиеся сценарии и пытается их выполнить. Если встретился вызов функции, то браузер ищет её описание в текущем и во всех ранее загруженных сценариях. Поэтому описание функции должно размещаться либо в одном сценарии с вызовом, либо в сценариях расположенных выше вызова функции.

Сценарий можно хранить в отдельном файле. Для вставки сценария в HTML-документ служит атрибут *SRC* тега <*SCRIPT*>**.** Сценарий, размещённый в отдельном файле, можно использовать на многих страницах сайта. В примере 1.2 рассматривается одна из страниц сайта сети магазинов.

Пример 1.2

```
html><!-- СКРИПТ загружается из файла primJs.js--> <HEAD>
<TITLE>Сеть</title>
</head>
<body>
<SCRIPT language="JavaScript" src="PrimJs.js">
</script>
<H2 align=center style="color:green">Магазин "ПОДАРКИ"</h2>
Адрес: Лесная ул., д.2<P>
Транспорт: трамваи 7, 23, автобусы 56, 93
</body>
</html>
// Файл primJs.js
a="background-color:#00ffff; color:#ff00ff;"
a+="font-size:24pt; font-family:'Times New Roman'"
naim='Сеть магазинов "ВСЁ ДЛЯ ДОМА"'
var da=new Date()
d=da.getDate()+"."+da.getMonth()+"."+da.getYear()
document.write('<P align=center style= "'+a+'">'+
naim+'</p><P>Сегодня '+d+'</p>')
```

Практическое задание.

1. Напишите HTML-документ, который в окне браузера отображается в виде следующих трёх строк:

- ДОСТУП К СВОЙСТВАМ И МЕТОДАМ
- Коллекция all
- Метод getElementById()

Первую строку поместите в контейнер <*H2*>...</*h2*>, вторую – в контейнер <*P*> ...</*p*>, третью – в контейнер <*DIV*> ... </*div*>. Напишите скрипт для изменения цветов

фона и букв надписей, при щелчке по этим строкам. При щелчке по первой строке цвет букв должен меняться с чёрного на белый или с белого на чёрный, а фон – с жёлтого на синий или с синего на жёлтый. Так же должны меняться цвета третьей строки.

При щелчках по второй строке цвет букв на ней должен меняться с красного на белый и наоборот, а цвет фона - с белого на зелёный и наоборот.

Для изменения первой и третьей строк примените метод *getElementById(),* а для второй строки – коллекцию *all*.

Лабораторная работа №19. Работа с элементами управления в Java Script.

Цель работы: научиться управлять формой через сценарий**.**

Теоретический материал.

Форма служит для ввода пользователем через окно браузера данных и передачи их на веб-сервер. Форма состоит из контейнера <FORM> …</form> и заключённых в него тегов (элементов) <*INPUT*>, <*SELECT*> и <*TEXTAREA*>.

Проверка данных перед отправкой на сервер

Для уменьшения нагрузки на сеть и веб-сервер можно проверять введённые данные на браузере с помощью сценария на JavaScript. Если в данных обнаружится ошибка, то пользователю предоставляется возможность её исправить. Введенные правильно данные отправляются на веб-сервер. Использование сценария для управления формой демонстрируется в примере 1.

Пример 1

```
<HTML>
<HEAD><TITLE>Первая страница</title></head>
<H2>Представьтесь, пожалуйста</h2>
<FORM name=F1 METHOD="POST" ACTION="">
Имя....
<INPUT TYPE="text" NAME="name"><BR>
Возраст <INPUT TYPE="text" NAME="age">
<P> <INPUT TYPE="submit" VALUE="ВВОД" onclick="Proverka()">
</FORM>
<SCRIPT>
function Proverka()
{ im=document.F1.name.value

vozr=document.forms[0].elements[1].value st=""
if(im == "") st="имя\n"
if(vozr == "") st+="возраст"
if(st == "") document.F1.action="Privet.php"
else //отмена передачи на веб-сервер
```

```
{   str="Введите:\n "+st
   alert(str)
   return=false
}

}
</script>
</html>

<HTML>
<HEAD><TITLE>Вторая страница</title></head>
<BODY>
<H3>П Р И В Е Т С Т В И Е</h3>
<?php
$imja=$_POST["name"]; //приём параметров из формы
$age=$_POST["age"]; $x="Здравствуйте!
$imja.";
if($age>50) echo "$x Вы включены в старшую группу.";
elseif($age>30)echo "$x Вы включеныв группу среднего возраста.";
else echo"$x Вы относитесь к молодёжной группе.";
?>
<P>
<A href="Form_Post.html">Возврат </a>
</body>
</html>
```

Пример 1 состоит из двух страниц. Первая страница служит для ввода пользователем данных и их проверки с помощью скрипта, написанного на JavaScript. Если данные введены правильно, то они отправляются на веб-сервер. На веб-сервере полученные данные обрабатываются скриптом, написанным на языке PHP, формируется и пересылается на браузер пользователя новая страница.

В скрипте,написанном на JavaScript, для доступа к данным, находящимся в форме, используются имена и индексы элементов формы. Для задания адреса (URL) страницы, содержащей PHP-скрипт, используется свойство action объекта *Form*. Отмена передачи данных из формы на веб-сервер делается оператором

event.returnValue= false

Пример 1 правильно выполняется в браузерах Internet Explorer 6.0 и Mozilla Firefox 2.0 .

Получение данных из всплывающего списка

Иногда можно полностью решать задачу ведения диалога с пользователем средствами JavaScript, не обращаясь к веб-серверу. В примере 3.2 пользователь вводит код цвета в модели RGB и выбирает из списка название цвета. После нажатия кнопки *Ввод* на экран выводятся окрашенные в выбранные цвета код и название цвета.

Пример 2

```
<html>
<HEAD><TITLE>СКИПТ SELECT </title> </head>
<body>
<!--Пример выбора и выводана экранэлемента списка Select -->
<SCRIPT>
function select_()
{ a=document.all.Kod.value //код цвета

b= document.all.Gor.selectedIndex
//номервыбранного элемента// списказelect
c = document.all.Gor.options[b].text//текст элементасписка
d = document.all.Gor.options[b].value//передаваемое из формы
//значение<option value= red>
e="<FONT size= 7 color="+a+">"+a+"</font>"//трансляция и
document.all.alfa.innerHTML= e// вывод на экран HTML-строки
e= "<FONT size = 7 color= "+d+">"+c+"</font>"
document.all.beta.innerHTML= e

}
</script>
<H2>Подбор оттенков цвета</h2>
В первое поле нужно ввести шестнадцатеричный код цвета.<BR>
Например,красный цвет имеет код FF0000.
<BR>Из списка во втором поле выбирается для сравнения
<BR> один из основных цветов(красный, зелёный,синий)
<P>Введитекод цвета
<input TYPE= "text"name="Kod"> <P>Выберитецвет
<SELECT NAME= "Gor">
<option value="red">Красный </option>
<option value="yellow">Жёлтый </option>
<option value="maroon">каштановый</option>
<option value=green">Зеленый</option>
<option value="blue">Синий
</select>
<P> <BUTTON onclick="select_()">Выполнить </button>
<P><B ID="alfa"></b>
<br> <B ID = "beta"></b>
</body>
</html>
```

В примере 2 нет необходимости использовать контейнер *<FORM> ...</form>*, так как на веб-сервер ничего не передаётся.

Проверка данных сразу после ввода

Если в форме нужно заполнить много полей, то пользователю удобно получать сообщения об ошибках сразу после окончания ввода данных в очередное поле, то есть

после нажатия клавиши *Tab* или клавиши со стрелкой. Для немедленной проверки введённых данных служит событие *onchange*:

<INPUT TYPE ="text"SIZE=6 onchange="arg(this)">

Функция, вызываемая событием *onchange*, имеет примерно такую структуру:

```
function arg(fld)
{ x=fld.value //введённое значение
  if(x. . .) //условия проверки
  { alert("Сообщение об ошибке");
    fld.focus();
    fld.select()
  }
}
```

Методы *focus()* и *select()* служат для возвращения курсора мышки в поле ввода и выделения ошибочных данных. Эти методы без использования специальных приёмов правильно работают только в браузере Mozilla.

Практическое задание.

Создайте сайт из двух страниц. Первая страница имеет заголовок *Заказ мебели*. На ней расположены два поля со списками (теги <*SELECT*>), поле *(<INPUT>)* и кнопка *(<SUBMIT>)*. Из первого поля со списком пользователь выбирает изделие (шкаф, стол, сервант и т.д.).Из второго поля со списком пользователь выбирает материал (дуб,орех, бук). В третье поле нужно ввести количество заказываемых изделий. После ввода данных необходимо проверить, все ли данные введены. Если обнаружена ошибка, то нужно вывести сообщение и предложить её исправить. Правильно введённые данные нужно отправить на веб-сервер. Вторая страница содержит написанный на PHPскрипт, с помощью которого формируется следующее сообщение:

Ваш заказ принят

Заказано изделие – *название заказанного изделия*

Материал – *заказанный материал*

Количество – *заказанное количество*

Лабораторная работа №20.
Реализация математических функций в Java Script.

Теоретический материал.

Арифметические функции и операторы в JavaScript

+ - сложение; - - вычитание; / - деление; * - умножение; % - остаток от деления.

parseInt - читает из строки целое число

parseFloat - читает из строки дроби

Проверка на число

Проверку типа данных на число можно сделать при помощи функции isNaN(), которая определяет является ли литерал или переменная нечисловым значением:

isNaN(10) // false

isNaN('строка') // true

Копировать

поэтому чтобы проверить значение на то является ли оно числом, перед isNaN ставят восклицательный знак !, котоый является логическим оператором НЕ (неравно, неправда):

!isNaN(10) // true

Копировать

Ещё один способ определить является ли значение числом - это использовать оператор typeof, который возвращает строку указывающую тип операнда:

console.log(typeof 23); // "number"

console.log(typeof 'abcde'); // "string"

console.log(typeof true); // "boolean"

let a = '23';

let b = 23;

let c = 'qwerty';

typeof a == 'number' // false

typeof b == 'number' // true

typeof c == 'number' // false

Копировать

Объект Math

Объект Math является встроенным объектом в язык JavaScript. Он хранит в своих свойствах и методах различные математические константы и функции. При этом объект Math не является функциональным объектом.

Math.ceil() - Округляет вверх

Math.floor() - Округляет вниз

Math.round() - Округляет до ближайшего целого

Math.trunc() - отрезает дробную часть и получается целое число. Например: Math.trunc(14,318) вернёт результат 14. Метод не округляет, а просто откидывает дробную часть.

.toFixed(2) - округляет число до 2 знаков. Цифра указывает сколько знаков оставлять после запятой.

Math.max() - возвращает самое большое число. Пример: Math.max(2, 73, 14, 47) вернёт 73.

Math.min() - возвращает наименьшее из чисел.

Работа со степенями

Math.sqrt() - Корень квадратный

```
let n = 25;

let koren = Math.sqrt(n); // 5
```

Math.cbrt() - Корень кубический

```
let n = 125;
let korenKub = Math.cbrt(n); // 5
```

Math.pow(num, st) - Возвести число в степень.

Аргументы:

- **num** - число которое возвести в степень,
- **st** - степень в которую возвести число.

```
let n = 5;
let kvadrat = Math.pow(n, 2);  // 25 квадрат числа
let kub = Math.pow(n, 3);      // 125 куб числа
```

Операторы сравнения

`>` - больше; `<` - меньше

`>=` - больше или равно; `<=` - меньше или равно

`==` - равно; `!=` - не равно

`!(x == y)` - не равно

`||` - оператор ИЛИ

```
if (x > 5 || y == 2)
```

Копировать

`&&` - оператор И

```
if (x > 5 && x <= 12)
```

Тригонометрия

`Math.sin(x)` - возвращает числовое значение от -1 до 1, которое представляет синус переданного (в радианах) угла

`Math.cos()` - возвращает косинус числа

`Math.tan()` - возвращает тангенс числа

`Math.acos()` - возвращает арккосинус числа

`Math.asin()` - возвращает арксинус числа

`Math.atan()` - возвращает арктангенс числа в радианах

`Math.atan2()` - возвращает арктангенс от частного своих аргументов

Рандомное число (случайное число)

Math.random() - Возвращает случайное число в диапазоне от 0 до 1.

Функция для генерации целых случайных чисел:

```
function getRandomInt(min, max) {
  return Math.floor(Math.random() * (max - min)) + min;
}
```

Копировать

Функция вовращает случайное целое число между min (включительно) и max (не включая max)

Детали реализации

• Определение кнопок через класс или id:

```
var name_variable = document.querySelectorAll(‘.name_class’),
name_element = document.getElementById(‘id_element’);
```

• Обработчики событий на кнопки:

```
1 name_variable.addEventListener(‘click’, name_eventHandler);
2 name_variable.addEventListener(‘click’, function (e) {
3 //обработка события
4 //e – элемент события
5 });
```

• Все свойства элемента можно вывести в консоль:

```
console.log(e);
```

Свойство для распознавания кнопки:

```
e.srcElement.id
```

Свойство для получения текста кнопки:

e.target.textContent

Общие структуры функций:

Функция, которая отражает введенное число на экране калькулятора:

```
function numberPress (number) {
if (условие) { //ввод нового значение и меняем значение флага}
else {
if (условие) { //стирает 0 с дисплея}
else { //добавляет цифру к числу на экране}
}
};
```

Функция обработки операций калькулятора и вывода результата:

```
function operation (op) {
//создаем локальную переменную памяти
if (условия) { //сохраняем значение на экране в переменную памяти}
else { //говорим переменной памяти о том, что мы вводим новое число}
if (условие с +) { //выполнение операции}
else if (условие с -) { //выполнение операции}
else if (условие с *) { //выполнение операции}
else if (условие с /) { //выполнение операции}
else { //действия с глобальной и локальной памятью};
//вывод результата
//сохранение текущей операции
};
```

Функция добавления десятичной точки:

```
function decimal (argument) {
```

```
//создаем локальную переменную
if (условие) { //добавляем 0.}
else {
if (условие) { //если не существует символа '.' в строке – добавляй точку };
};
};
```

Функция обработки кнопок очищения C и CE:

```
function clear (argument) {
if (условие с ce) { //удаление введённого числа с дисплея}
else if (условие с c) { //очищения дисплея };
};
```

Функция вывода списка функций калькулятора — «Как это работает?»

```
function howWork (argument) {
//новый элемент списка
for (по операциям) {
//добавление в элемент списка текста
//вставка в лист элемента
};
};
```

Практическое задание.

1. Разработать дизайн калькулятора, используя средства HTML и CSS (при желании).
2. Подключить к его элементам математические функции.

Лабораторная работа №21. Настройка веб-сервера и связи с базой данных

Теоретический материал.

Что нужно для начала работы с PHP?

Нам понадобится установить себе на компьютер самый настоящий **веб-сервер**, который, между прочим, тоже бесплатный, а называется он **Apache**. Но это ещё не всё, после установки сервера Apache нам ещё придётся прикрутить к нему **модуль PHP**. И в этом нам поможет бесплатный дистрибутив разработанный Дмитрием Котеровым под названием **Denwer (Денвер) - Джентльменский набор Web-разработчика**.

Данный дистрибутив (приложение), имеющий стандартное расширение для Windows .exe, на данный момент включает в себя Apache 2.2.22 + SSL, PHP 5.3.13 + XDebug, MySQL 5.5, phpMyAdmin 3.5. Скачать Денвер с официального сайта вы можете по этой **ссылке**, а прочитать подробную инструкцию по его установке можно **здесь**.

После установки Денвера вы будете обладать всем необходимым ПО (программным обеспечением), которое нужно для разработки сайтов на PHP, а также для установки и использования любой CMS (Joomla, Wordpress и тд.).

Также подразумевается, что у вас уже есть хотя бы базовые познания в области **HTML + CSS**, а, следовательно, вы уже успели хотя бы чуть-чуть поработать с такими инструментами для разработки сайтов как **веб-инспекторы** и уж точно знаете **из чего состоит сайт**. Ещё стоит отметить, что работу PHP разработчика может заметным образом облегчить правильно подобранный редактор кода, в котором вы быстро освоитесь и будете использовать все встроенные возможности редактора по максимуму и с наибольшим КПД. Примером такого редактора, кстати, абсолютно бесплатного, может послужить **Sublime Text 2**. Перейдя по ссылке, вы можете прочитать о нём обзорную статью и научиться некоторым тонкостям работы.

Начало работы с PHP на Денвере

Итак, чтобы запустить нашу первую веб-страницу со встроенным PHP скриптом необходимо обязательно поменять расширение нашего файла с .html на .php У меня он будет называться как ни странно index.php Располагать его требуется в папке с Денвером по следующему пути:

WebServers\home\localhost\www\название_папки_с_вашим_сайтом(произвольное)

Запустить такой файл простым перетягиванием в браузер, как мы делали это раньше, не получится. Сначала нам нужно не забыть сделать двойной клик по «Start Denwer», чтобы запустить наш веб-сервер Apache, а затем в адресную строку браузера ввести следующий url:

http://localhost/blog2/index.php

Где каталог blog2 – это та самая папка с произвольным названием для вашего сайта (измените на свою).

Получилось? Тогда рад вас поздравить, вы только что создали свою первую веб-страницу с PHP вставками. На сегодня всё, в следующих статьях мы начнём уже подробное изучение скриптового языка программирования – PHP.

PHP поддерживает работу с базой данных MySQL.

Специальные встроенные функции для работы с MySQL позволяют просто и эффективно работать с этой СУБД: выполнять любые запросы, читать и записывать данные, обрабатывать ошибки.

Сценарий, который подключается к БД, выполняет запрос и показывает результат, будет состоять всего из нескольких строк. Для работы с MySQL не надо ничего дополнительно устанавливать и настраивать; всё необходимое уже доступно вместе со стандартной поставкой PHP.

Что такое mysqli?

mysqli (MySQL Improved) — это расширение PHP, которое добавляет в язык полную поддержку баз данных MySQL. Это расширение поддерживает множество возможностей современных версий MySQL.

Как выглядит работа с базой данных

Типичный процесс работы с СУБД в PHP-сценарии состоит из нескольких шагов:

1. Установить подключение к серверу СУБД, передав необходимые параметры: адрес, логин, пароль.
2. Убедиться, что подключение прошло успешно: сервер СУБД доступен, логин и пароль верные и так далее.
3. Сформировать правильный SQL запрос (например, на чтение данных из таблицы).
4. Убедиться, что запрос был выполнен успешно.
5. Получить результат от СУБД в виде массива из записей.
6. Использовать полученные записи в своём сценарии (например, показать их в виде таблицы).

Функция mysqli connect: соединение с MySQL

Перед началом работы с данными внутри MySQL, нужно открыть соединение с сервером СУБД.

В PHP это делается с помощью стандартной функции `mysqli_connect()`. Функция возвращает результат — ресурс соединения. Данный ресурс используется для всех следующих операций с MySQL.

Но чтобы выполнить соединение с сервером, необходимо знать как минимум три параметра:

- Адрес сервера СУБД;
- Логин;
- Пароль.

Если вы следовали стандартной процедуре установки MySQL или используете OpenServer, то адресом сервера будет `localhost`, логином — `root`. При использовании OpenServer пароль для подключения — это пустая строка ‘’, а при самостоятельной установке MySQL пароль вы задавали в одном из шагов мастера установки.

Базовый синтаксис функции `mysqli_connect()`:

```
mysqli_connect(<адрес сервера>, <имя пользователя>, <пароль>, <имя базы данных>);
```

Проверка соединения

Первое, что нужно сделать после соединения с СУБД — это выполнить проверку, что оно было успешным.

Эта проверка нужна, чтобы исключить ошибку при подключении к БД. Неверные параметры подключения, неправильная настройка или высокая нагрузка заставит MySQL отвергать новые подключения. Все эти ситуации приведут к невозможности соединения, поэтому программист должен проверить успешность подключения к серверу, прежде чем выполнять следующие действия.

Соединение с MySQL устанавливается один раз в сценарии, а затем используется при всех запросах к БД.

Результатом выполнения функции `mysqli_connect()` будет значение специального типа — ресурс.

Если подключение к MySQL не удалось, то функция `mysqli_connect()` вместо ресурса вернет логическое значение типа «ложь» — `false`.

Хорошей практикой будет всегда проверять значение результа выполнения этой функции и сравнивать его с ложью.

Соединение с MySQL и проверка на ошибки:

```
<?php
$link = mysqli_connect("localhost", "root", "");

if ($link == false){
  print("Ошибка: Невозможно подключиться к MySQL " . mysqli_connect_error());
}
else {
  print("Соединение установлено успешно");
}
```

Функция `mysqli_connect_error()` просто возвращает текстовое описание последней ошибки MySQL.

Установка кодировки

Первым делом после установки соединения крайне желательно явно задать кодировку, которая будет использоваться при обмене данными с MySQL. Если этого не сделать, то вместо записей со значениями, написанными кириллицей, можно получить последовательность из знаков вопроса: '??????????????????'.

Вызови эту функцию сразу после успешной установки соединения: mysqli_set_charset($con, "utf8");

Выполнение запросов

Установив соединение и определив кодировку мы готовы выполнить свои первые SQL-запросы. Вы уже умеете составлять корректные SQL команды и выполнять их через консольный или визуальный интерфейс MySQL-клиента.

Те же самые запросы можно отправлять без изменений и из PHP-сценария. Помогут в этом несколько встроенных функций языка.

Два вида запросов

Следует разделять все SQL-запросы на две группы:

1. Чтение информации (SELECT).
2. Модификация (UPDATE, INSERT, DELETE).

При выполнении запросов из среды PHP, запросы из второй группы возвращают только результат их исполнения: успех или ошибку.

Запросы первой группы при успешном выполнении возвращают специальный ресурс результата. Его, в свою очередь, можно преобразовать в ассоциативный массив (если нужна одна запись) или в двумерный массив (если требуется список записей).

Добавление записи

Вернёмся к нашему проекту — дневнику наблюдений за погодой. Начнём практическую работу с заполнения таблиц данными. Для начала добавим хотя бы один город в таблицу cities.

Выражение `INSERT INTO` используется для добавления новых записей в таблицу базы данных.

Составим корректный SQL-запрос на вставку записи с именем города, а затем выполним его путём передачи этого запроса в функцию `mysqli_query()`, чтобы добавить новые данные в таблицу.

```
<?php
$link = mysqli_connect("localhost", "root", "");
```

```
$sql = 'INSERT INTO cities SET name = "Санкт-Петербург"';
$result = mysqli_query($link, $sql);

if ($result == false) {
   print("Произошла ошибка при выполнении запроса");
}
```

Обратите внимание, что первым параметром для функциии `mysqli_query()` передаётся ресурс подключения, полученный от функции `mysqli_connect()`, вторым параметром следует строка с SQL-запросом.

При запросах на изменение данных (не SELECT) результатом выполнения будет логическое значение — true или false.

`false` будет означать, что запрос выполнить не удалось. Для получения строки с описанием ошибки существует функция `mysqli_error($link)`.

Функция insert id: как получить идентификатор добавленной записи

Следующим шагом будет добавление погодной записи для нового города. Погодные записи хранит таблица weather_log, но, чтобы сослаться на город, необходимо знать идентификатор записи из таблицы cities.

Здесь пригодится функция `mysqli_insert_id()`.

Она принимает единственный аргумент — ресурс соединения, а возвращает идентификатор последней добавленной записи.

Теперь у нас есть всё необходимое, чтобы добавить погодную запись.

Вот как будет выглядеть комплексный пример с подключением к MySQL и добавлением двух новых записей:

```
<?php
$link = mysqli_connect("localhost", "root", "");

if ($link == false){
   print("Ошибка: Невозможно подключиться к MySQL " . mysqli_connect_error());
}
else {
   $sql = 'INSERT INTO cities SET name = "Санкт-Петербург"';
   $result = mysqli_query($link, $sql);

   if ($result == false) {
      print("Произошла ошибка при выполнении запроса");
   }
```

```
  else {
    $city_id = mysqli_insert_id($link);

    $sql = 'INSERT INTO weather_log SET city_id = ' . $city_id . ', day = "2017-09-03",
temperature = 10, cloud = 1';

    $result = mysqli_query($link, $sql);

    if ($result == false) {
      print("Произошла ошибка при выполнении запроса");
    }
  }
}
```

Чтение записей

Другая частая операция при работе с базами данных в PHP — это получение записей из таблиц (запросы типа SELECT).

Составим SQL-запрос, который будет использовать `SELECT` выражение. Затем выполним этот запрос с помощью функции `mysqli_query()`, чтобы получить данные из таблицы.

В этом примере показано, как вывести все существующие города из таблицы cities:

```
<?php

$sql = 'SELECT id, name FROM cities';

$result = mysqli_query($link, $sql);

while ($row = mysqli_fetch_array($result)) {
  print("Город: " . $row['name'] . "; Идентификатор: . " . $row['id'] . "<br>");
}
```

В примере выше результат выполнения функции mysqli_query() сохранён в переменной $result.

Важно понимать, что в этой переменной находятся не данные из таблицы, а специальный тип данных — так называемая ссылка на результаты запроса.

Чтобы получить действительные данные, то есть записи из таблицы, следует использовать другую функцию — mysqli_fetch_array() — и передать ей единственным параметром эту самую ссылку.

Теперь каждый вызов функции mysqli_fetch_array() будет возвращать следующую запись из всего результирующего набора записей в виде ассоциативного массива.

Цикл while здесь используется для «прохода» по всем записям из полученного набора записей.

Значение поля каждой записи можно узнать, просто обратившись по ключу этого ассоциативного массива.

Как получить сразу все записи в виде двумерного массива

Иногда бывает удобно после запроса на чтение не вызывать в цикле mysqli_fetch_arrayдля извлечения очередной записи по порядку, а получить их сразу все одним вызовом. PHP так тоже умеет. Функция mysqli_fetch_all($res, MYSQLI_ASSOC) вернёт двумерный массив со всеми записями из результата последнего запроса.

Перепишем пример с показом существующих городов с её использованием:

```
<?php

$sql = 'SELECT id, name FROM cities';
$result = mysqli_query($link, $sql);

$rows = mysqli_fetch_all($result, MYSQLI_ASSOC)

foreach ($rows as $row) {
   print("Город: " . $row['name'] . "; Идентификатор: . " . $row['id'] . "<br>");
}
```

Как узнать количество записей

Часто бывает необходимо узнать, сколько всего записей вернёт выполненный SQL запрос. Это может помочь при организации постраничной навигации, или просто в качестве информации.

Узнать число записей поможет функция mysqli_num_rows(), которой следует передать ссылку на результат запроса.

Практическое задание.

1. Установите на рабочем ПК веб-сервер Apache или XAMPP, чтобы PHP-код работал.
2. Напишите любой PHP-код в составе стандартной HTML-страницы, проверьте его работоспособность.
3. Реализуйте визуальное приложение для работы с вашими базами данных, которые вы используете для других междисциплинарных комплексов.
4. Осуществите подключение к этим базам данных.

Лабораторная работа №22. Использование управляющих структур и функций PHP

Теоретический материал.

Все скрипты в PHP представляют собой набор различных выражений, которые выполняются последовательно. Выражения можно объединять в группы выражений при

помощи т.н. "операторных скобок" "{" и "}". Группы выражений используются в основном вместе с управляющими конструкциями языка PHP.

Управляющие конструкции языка - это наборы служебных слов, позволяющие изменять ход выполнения скрипта. Все конструкции можно условно разделить на конструкции бинарного выбора, множественного выбора, повторения и включения.

Конструкции бинарного (двойственного) выбора позволяют в зависимости от условия выполнить либо первое, либо второе действие. В PHP эти конструкции представлены ключевыми словами if, else, elseif и endif.

Конструкция if позволяет выполнить какое-то действие, если условие истинно. Этот пример читается как "ЕСЛИ условие истинно ТО выполнить выражения".

```
<?php
  if(условие)
    одиночное_выражение;
  // или
  if(условие)
  {
    несколько;
    последовательных;
    выражений;
  }
?>
```

Конструкция else используется совместно с if и определяет действие, когда условие ложно. Нижеследующий пример читается как "ЕСЛИ условие истинно ТО выполнить выражения-1 ИНАЧЕ выполнить выражения-2".

```
<?php
  if(условие)
  {
    выражения-1;
```

```
  }
  else
  {
    выражения-2;
  }
?>
```

Конструкция elseif аналогична else, но позволяет создавать цепочки условий и действий. Приведённый ниже пример читается как "ЕСЛИ условие-1 истинно ТО выполнить выражения-1 ИНАЧЕ ЕСЛИ условие-2 истинно ТО выполнить выражения-2 ИНАЧЕ ...".

```
<?php
  if(условие-1)
  {
    выражения-1;
  }
  elseif(условие-2)
  {
    выражения-2;
  }
  elseif(условие-3)
  ...

?>
```

Конструкция endif может использоваться в случае альтерантивной формы записи управляющих конструкций. Отличие альтернативной формы от стандартной в том, что открывающая операторная скобка "{" заменяется на двоеточие ":", закрывающая скобка удалятся, а граница конструкции определяется по ключевому слову endXXX, где XXX -

тип конструкции. Например, для связки if...endif...else тип конструкции - "if", значит граница будет определяться по ключевому слову "endif". Сравните:

```
<?php
   // Стандартная форма записи:
   if(условие-1)
   {
     выражения-1;
   }
   elseif(условие-2)
   {
     выражения-2;
   }
   else
   {
     выражения-3;
   }

   // Альтернативная форма записи:
   if(условие-1):
     выражения-1;
   elseif(условие-2):
     выражения-2;
   else:
     выражения-3;
   endif;
```

```
?>
```

Какой из форм пользоваться - каждый разработчик решает для себя сам, но большинство всё же придерживается стандартной формы записи (т.е. с нормальными операторными скобками "{" и "}")

Конструкция множественного выбора представляет собой компактную форму записи длинных цепочек условий вида "if...elseif....elseif.....else". В PHP такая конструкция носит название switch и имеет достаточно простую форму записи:

```
<?php
  switch (выражение/переменная)
  {
     case значение-1:
        выражения-1;
        break;
     case значение-2:
        выражения-2;
        break;
     ...

     default:
        выражения-N;
        break;
  }
?>
```

Т.е. мы указываем переменную или выражение, после чего указываем возможные значения и действия, соответствующие этим значениям. Все значения, которые мы явно не указали, попадут в блок "default". Замечательная особенность switch - возможность

указывать список выражений одновременно для нескольких вариантов. Для этого несколько блоков "case" записываются последовательно, например:

```
<?php
   switch (выражение/переменная)
   {
      case значение-1:
      case значение-2:
      case значение-3:
      case значение-4:
         выражения_для_случаев_1_2_3_4;
         break;
      case значение-5:
         выражения_для_случая_5;
         break;
      ...
      default:
         выражения-N;
         break;
   }
?>
```

Switch очень удобен, когда значения принимают конкретные фиксированные значения. Например, действия водителя на светофоре можно записать так:

```
<?php
   $color = 'red';

   switch ($color)
```

```
{
    case 'красный':
        echo 'Стоять';
        break;

    case 'красно-желтый':
        echo 'Приготовиться';
        break;

    case 'желтый':
        echo 'Остановиться';
        break;

    case 'желтый мигающий':
        echo 'По обстановке';
        break;

    case 'зелёный':
        echo 'Ехать';
        break;

    default:
        // например, светофор выключен
        echo 'По обстановке';
        break;
```

```
    }
?>
```

Конструкции повторения (организации циклов) предназначены для многократного выполнения одних и тех же выражений. К этим конструкциям относятся while, do-while, for и foreach.

While ("уайл") и do-while ("ду-уайл") предназначены для организации циклов в случаях, когда число повторений заранее неизвестно или может измениться в процессе выполнения цикла. Основой этих конструкций является логическое выражение (условие), а цикл повторяется до тех пор, пока условие истинно (равно TRUE).

Формальная запись конструкций выглядит так:

```
<?php
    while (условие)
    {
        выражения;
    }

    do
    {
        выражения;
    }
    while (условие);
?>
```

Главное отличие while от do-while в том, что do-while всегда выполняется хотя бы один раз даже если условие изначально ложно. Например:

```
<?php
    $i = 0;
    while ($i > 0)
```

```
    {
        echo 'i больше нуля! (цикл while)';
    }
    // не выводится ничего

    do
    {
        echo 'i больше нуля! (цикл do-while)';
    }
    while ($i > 0);

    // выводится "i больше нуля! (цикл do-while)"
?>
```

Из примера видно, что пользоваться циклом do-while нужно аккуратно, иначе есть большой риск получить неожиданные и парадоксальные результаты, такие как в нашем примере. Неверное применение циклов очень часто приводит к повялению логических ошибок, которые на порядок сложнее выявить и локализовать, нежели все прочие.

Конструкции while и do-while очень удобно использовать при построчном чтении из файла или обработке результатов запросов к базам данных, т.к. заранее неизвестно, сколько раз надо будет выполнить требуемый набор выражений.

```
<?php
    $i = 0;
    while ($i > 0)
    {
        echo 'i больше нуля! (цикл while)';
    }
    // не выводится ничего
```

```
   do
   {
      echo 'i больше нуля! (цикл do-while)';
   }
   while ($i > 0);

   // выводится "i больше нуля! (цикл do-while)"
?>
```

Следующая конструкция - for ("фор"). Эта конструкция предполагает, что количество итераций цикла заранее известно или вычислимо до начала цикла. Формально for записывается так:

```
<?php
   for (переменная; условие; оператор )
   {
      выражения;
   }
?>
```

Блок "переменная" описывает переменную-счётчик и её начальное значение, "условие" определяет количество итераций, а "оператор" - действие над переменной-счётчиком ПОСЛЕ каждой итерации. Например, если вам надо вывести 10 раз одну и ту же строку, можно воспользоваться таким кодом:

```
<?php
   for ($i = 0; $i < 10; $i++ )
   {
      echo "Копия строки номер " . $i;
   }
```

```
?>
```

Здесь мы указали, что $i - это переменная счётчик, считать начинаем с нуля, после каждой итерации учеличиваем $i на единицу, продолжаем итерации пока $i меньше 10. Приведённй код цикла функционально эквивалентен следующему (записанному без цикла):

```
<?php
   echo "Копия строки номер 0";
   echo "Копия строки номер 1";
   ...
   echo "Копия строки номер 9";
?>
```

Для цикла for совершенно не обязательно знать количество итераций на этапе написания скрипта. Главное - мы должны суметь вычислить это количество и передать в цикл. Например:

```
<?php
   // код можно использовать, например, для
   // создания навигатора по
   // страницам каталога товаров

   // общее количество элементов в каталоге
   $rows = 100;

   // количество одновременно отображаемых
   // элементов на странице
   $rows_per_page = 16;

   // вычисляем количество страниц
```

```
    $pages = ceil($rows/$rows_per_page);


    $page_line = '';


    // для каждой страницы создаём соответствующую ссылку
    for ($i = 0; $i < $pages; $i++ )
    {
        $page_line .= '<a href="?page=' . $i .
                '">[' . ($i + 1) . ']</a>';
    }


    // выводим готовую строку навигации
    echo $page_line;
    // выведет "[1][2][3][4][5][6][7]";
?>
```

Последняя конструкция циклов - foreach. Это самая простая и самая своеобразная конструкция из рассмотренных. Записывается она так:

```
<?php
    foreach(массив as формат_элемента)
    {
        выражения;
    }
?>
```

Здесь "массив" - переменная типа array, созданная ранее, а "формат_элемента" - это формальное описание одного элемента массива. Рассмотрим на примерах:

```
<?php
```

```
// одномерный не-ассоциативный массив
$names = array('Александр', 'Владимир', 'Ярослав');

foreach($names as $single_name)
{
    echo 'имя из массива:' . $name . '<br />\n';
}

// одномерный ассоциативный массив
$names = array(
        'Имя' => 'Александр',
        'Фамилия' => 'Владимирович',
        'Отчество' => 'Генералов',
        'Телефон' => '+7(123)456-78-90');

foreach($names as $param => $value)
{
    echo $param . ': ' . $value . '<br />\n';
}

?>
```

При использования многомерных массивов в foreach они будут интерпретироваться как вложенные ассоциативные массивы. Если в последнем примере $names будет двумерным массивом, то $param будет содержать номер массива перового уровня, а $value - массив второго уровня. Подробнее о массивах и работе с ними мы поговорим в соответствующем уроке.

Цикл foreach перебирает все элементы массива, независимо от их количества. Недостаток этой конструкции - невозможность модификации элементов массива во время итерации. Связано это с тем, что foreach перед началом цикла создаёт копию массива, которая уничтожается после окончания цикла. Следовательно, все изменения при выходе из цикла теряются. Этот недостаток был исправлен в PHP версии 5 и выше за счет добавления ссылки на элемент данных:

```
<?php
   // одномерный не-ассоциативный массив
   $names = array('Александр', 'Владимир', 'Ярослав');

   foreach($names as &$single_name)
   {
      $name .= 'ович';
   }

   echo implode(', ', $names);
   // в PHP до 5 версии выведет:
   // "Александр, Владимир, Ярослав"
   // в PHP 5 выведет:
   // "Александрович, Владимирович, Ярославович"
?>
```

Специально для управления исполнением циклов и скриптов в PHP существует несколько ключевых слов: break, continue и return.

break применяется внутри циклов и служит для немендленного прекращения итераций цикла. Управление передаётся на следующее после цикла выражение. Например, можно прервать цикл при возникновении определённых условий:

```
<?php
   $names = ('Ярослав', 'Александр', ...);
```

```
    foreach($i = 0; $i < 100; $i++)
    {
        if($names[$i] == 'Александр') break;
    }
    echo 'Имя "Александр" стоит на ' . $i . ' позиции';

    // выведет:
    // 'Имя "Александр" стоит на 2 позиции'
?>
```

break очень полезен в случае поиска нужного элемента массива, т.к. можно остановить поиск сразу после нахождения нужного элемента и не лопатить заведомо пустой остаток массива.

Следующее ключевое слово - continue. Предназначено для немедленного перехода к следующей итерации. Например, можно преобразовать предыдущий пример следующим образом:

```
<?php
    $names = ('Ярослав', 'Александр', ...);
    foreach($i = 0; $i < 100; $i++)
    {
        if($names[$i] != 'Александр') continue;
        echo 'Имя "Александр" стоит на ' . $i . ' позиции';
    }

    // выведет:
    // 'Имя "Александр" стоит на 2 позиции'
?>
```

Такой цикл переберёт все элементы массива и выполнит код "echo ..." только для тех элементов, где значение равно "Александр". Такое построение цикла удобно применять для случаев, когда выражения должны быть применены к нескольким элемента массива.

Последнее ключевое слово - return. Оно предназначено для немедленного выхода из функции и возврата значения (при необходимости). Подробнее об использовании return мы поговорим при рассмотрении функций.

Последняя группа конструкций - конструкции включения. Они предназначены для включения в текст скрипта каких-либо данных и кода, находящихся в другом файле. Всего существует четыре варианта: include, include_once, require, require_once.

Конструкции с приставкой "_once" отличаются от прочих тем, что гарантируют однократное включение файла в рамках всех задействованых файлов. Т.е. если у вас есть 10 файлов и в каждом необходимо подключить файл с описанием, например, класса, то при использовании обычных require или include PHP выдаст ошибку "недопустимо повторное обявление класса" уже на второй встретившейся команде включения. С другой стороны, include_once или require_once предварительно проверят предыдущие включения, и если файл уже был подключен - повторно подключать его не будут.

Отличие между include и require заключено в поведении при отсутсвующем файле для подключения. Если include или inclide_once не находят указанный файл, то выдают предупреждение для пользователя. А вот require и require_once генерируют ошибку и прекращают дальнейшее выполнение скрипта.

Формат всех четырёх конструкций практически одинаков. Имена подключаемых файлов можно записывать как в полной, так и сокращённой форме.

```
<?php

   include('engine/filename.php');

   include_once('myclass.class.php');

   require('./../utils/db.php');

   require_once('./../config/config_db.inc');


?>
```

Я бы рекомендовал максимально часто пользоваться require_once с относительными путями к файлам. Во-первых, это гарантирует наличие всех требуемых файлов, а во-вторых - делает невозможными ошибки повторного объявления классов и переменных. А в третьих - увеличивает гибкость и переносимость кода. В некоторых случаях грамотно построеная система подключаемых файлов значительно экономит силы и время разработчика, т.к. позволяет повторно использовать многие фрагменты кода.

Практическое задание.

1. Напишите код, который решает следующую математическую задачу: Заданы координаты трех вершин треугольника (x1, y1), (x2, y2) и (x3, y3). Найти его периметр и площадь.
2. Напишите код, который решает следующую математическую задачу: Найти max{min(a,b), min(c,d)}.
3. Напишите код, который решает следующую математическую задачу: Вычислить длину окружности и площадь круга одного и того же заданного радиуса R.
4. Напишите код, который решает следующую математическую задачу: Среди всех n-значных чисел указать те, сумма цифр которых равна данному числу k.
5. Напишите код, который решает следующую математическую задачу: Составить программу, которая печатает таблицу умножения.
6. Напишите код, который решает следующую математическую задачу: Подсчитать количество отрицательных чисел среди чисел a, b, c.