Дисциплина: **Проектирование информационных систем**

специальность: **информатики**

курс, группа: **3 к 1 г**

форма обучения: **заочная**

период проведения занятий, вид занятий, кол-во часов:

-**31 октября 2020 г.**: Лекции 4 часа, оф-лайн;

преподаватель: **Астахов В.К.**

электронная почта преподавателя: **vadast@mail.ru**

**Литература:**

## 1 Основная литература:

1. Грекул, В. И. Проектирование информационных систем. Курс лекций: учебное пособие для студентов вузов, обучающихся по специальностям в области информационных технологий / В. И. Грекул, Г. Н. Денищенко, Н. Л. Коровкина. — Москва, Саратов: Интернет-Университет Информационных Технологий (ИНТУИТ), Вузовское образование, 2017. — 303 c. — ISBN 978-5-4487-0089-7. — Текст: электронный // Электронно-библиотечная система IPR BOOKS: [сайт]. — URL: http://www.iprbookshop.ru/67376.html (дата обращения: 17.11.2019). — Режим доступа: для авторизир. пользователей
2. Коцюба, И. Ю. Основы проектирования информационных систем: учебное пособие / И. Ю. Коцюба, А. В. Чунаев, А. Н. Шиков. — СПб.: Университет ИТМО, 2015. — 205 c. — ISBN 2227-8397. — Текст: электронный // Электронно-библиотечная система IPR BOOKS: [сайт]. — URL: http://www.iprbookshop.ru/67498.html (дата обращения: 17.11.2019). — Режим доступа: для авторизир. пользователей
3. Проектирование информационных систем. Проектирование информационных систем: учебное пособие для студентов дневного и заочного отделений, изучающих курсы «Проектирование информационных систем», «Проектирование информационных систем», обучающихся по направлению 230700.62 (09.03.03) / А. В. Платёнкин, И. П. Рак, А. В. Терехов, В. Н. Чернышов. — Тамбов: Тамбовский государственный технический университет, ЭБС АСВ, 2015. — 80 c. — ISBN 978-5-8265-1409-2. — Текст: электронный // Электронно-библиотечная система IPR BOOKS: [сайт]. — URL: http://www.iprbookshop.ru/64560.html (дата обращения: 17.11.2019). — Режим доступа: для авторизир. пользователей (гриф)

## 2. Дополнительная литература:

1. Антонов, В. Ф. Методы и средства проектирования информационных систем: учебное пособие / В. Ф. Антонов, А. А. Москвитин. — Ставрополь: Северо-Кавказский федеральный университет, 2016. — 342 c. — ISBN 2227-8397. — Текст: электронный // Электронно-библиотечная система IPR BOOKS: [сайт]. — URL: http://www.iprbookshop.ru /66080.html (дата обращения: 17.11.2019). — Режим доступа: для авторизир. пользователей
2. Информационные системы и технологии в экономике и управлении. Проектирование информационных систем [Электронный ресурс]: учебное пособие / Е. В. Акимова, Д. А. Акимов, Е. В. Катунцов, А. Б. Маховиков. — Электрон. текстовые данные. — Саратов: Вузовское образование, 2016. — 178 c. — 2227-8397. — Режим доступа: http://www.iprbookshop.ru/47671.html
3. Крахоткина, Е. В. Методы и средства проектирования информационных систем и технологий [Электронный ресурс]: учебное пособие / Е. В. Крахоткина. — Электрон. текстовые данные. — Ставрополь: Северо-Кавказский федеральный университет, 2015.— 152 c. — 2227-8397. — Режим доступа: http://www.iprbookshop.ru/62959.html

**Интернет- ресурсы**

1. Библиотека публикаций и форумы по разработке и применению информационных систем [Электронный ресурс]: раздел Информационные технологии. — Электрон. дан. — Режим доступа: http://www. sql.ru
2. Библиотека публикаций по применению систем управления бизнес–процессами [Электронный ресурс]: раздел Информационные технологии. — Электрон. дан. — Режим доступа: http://www. bpms.ru
3. Бычков А.В. Проектирование информационных систем/Учебное пособие [Электронный ресурс]: офиц.сайт. — Электрон. дан. — Режим доступа: http://www.[twirpx.com](http://www.twirpx.com/)
4. Научная электронная библиотека eLIBRARY.ru [Электронный ресурс]: раздел Информационные технологии. — Электрон. дан. — Режим доступа: http://www.elibrary.ru/ defaultx.asp
5. Научная электронная библиотека IPRbooks.ru [Электронный ресурс]: раздел Информационные технологии. — Электрон. дан. — Режим доступа: http://www. iprbooks.ru (по паролю)
6. Научная электронная онлайн-библиотека Порталус [Электронный ресурс]: раздел Информационные технологии. — Электрон. дан. — Режим доступа: http://www. portalus.ru
7. Практические задания по проектированию информационных систем [Электронный ресурс]: офиц.сайт. — Электрон. дан. — Режим доступа: http://www. nto.immpu.sgu.ru
8. Проектирование информационных систем: видео лекции  [Электронный ресурс]: офиц.сайт. — Электрон. дан. — Режим доступа: http://www. niu.ranepa.ru
9. Проектирование информационных систем: видео лекции  [Электронный ресурс]: офиц.сайт. — Электрон. дан. — Режим доступа: http://www.edu.jobsmarket.ru/
10. Проектирование информационных систем: видео пособие, видео лекции  [Электронный ресурс]: офиц.сайт. — Электрон. дан. — Режим доступа: http://www.video. yandex.ru
11. Проектирование информационных систем: видео пособие, видео лекции [Электронный ресурс]: офиц.сайт. — Электрон. дан. — Режим доступа: http://www.video.yandex.ru

**Задания по темам и датам**

| **дата, время**  **занятия** | **вид**  **занятия** | **кол-во часов** | **вопросы для изучения и обсуждения** | **контрольные вопросы, задания** |
| --- | --- | --- | --- | --- |
| 31.10.2020  13.00-14.30 | Лекция 1 | 2 | 1.Средства проектирования, основанные на CASE-технологиях | 1.Изучить электронный конспект лекций и рекомендованную литературу по вопросам занятия.  (присылать материалы преподавателю для проверки НЕ НУЖНО) |
| 31.10.2020  14.35-16.05 | Лекция 2 | 2 | 2.Планирование и оценка проекта | 1.Изучить электронный конспект лекций и рекомендованную литературу по вопросам занятия.  (присылать материалы преподавателю для проверки НЕ НУЖНО) |

**Рекомендации и требования к выполнению заданий**

Изучить теоретический материал. Для подготовки использовать рекомендованную литературу и ниже представленный теоретический материал. Дополнительно отчитываться и присылать материал преподавателю не надо.

**Лекция 1. Средства проектирования, основанные на CASE-технологиях**

**1. Моделирование данных**

**Case-метод Баркера**

Цель моделирования данных состоит в обеспечении разработчика ИС концептуальной схемой базы данных в форме одной модели или нескольких локальных моделей, которые относительно легко могут быть отображены в любую систему баз данных.

Наиболее распространенным средством моделирования данных являются диаграммы "сущность-связь" (ERD). С их помощью определяются важные для предметной области объекты (сущности), их свойства (атрибуты) и отношения друг с другом (связи). ERD непосредственно используются для проектирования реляционных баз данных.

Нотация ERD была впервые введена П. Ченом (Chen) и получила дальнейшее развитие в работах Баркера [8]. Метод Баркера будет излагаться на примере моделирования деятельности компании по торговле автомобилями. Ниже приведены выдержки из интервью, проведенного с персоналом компании.

Главный менеджер: одна из основных обязанностей - содержание автомобильного имущества. Он должен знать, сколько заплачено за машины и каковы накладные расходы. Обладая этой информацией, он может установить нижнюю цену, за которую мог бы продать данный экземпляр. Кроме того, он несет ответственность за продавцов и ему нужно знать, кто что продает и сколько машин продал каждый из них.

Продавец: ему нужно знать, какую цену запрашивать и какова нижняя цена, за которую можно совершить сделку. Кроме того, ему нужна основная информация о машинах: год выпуска, марка, модель и т.п.

Администратор: его задача сводится к составлению контрактов, для чего нужна информация о покупателе, автомашине и продавце, поскольку именно контракты приносят продавцам вознаграждения за продажи.

Первый шаг моделирования - извлечение информации из интервью и выделение сущностей.

**Сущность (Entity)** - реальный либо воображаемый объект, имеющий существенное значение для рассматриваемой предметной области, информация о котором подлежит хранению (рисунок 1).

http://gendocs.ru/docs/1/395/conv_1/file1_html_m78fa3b6.png  
  
*Рис.1. Графическое изображение сущности*

Каждая сущность должна обладать уникальным идентификатором. Каждый экземпляр сущности должен однозначно идентифицироваться и отличаться от всех других экземпляров данного типа сущности. Каждая сущность должна обладать некоторыми свойствами:

* каждая сущность должна иметь уникальное имя, и к одному и тому же имени должна всегда применяться одна и та же интерпретация. Одна и та же интерпретация не может применяться к различным именам, если только они не являются псевдонимами;
* сущность обладает одним или несколькими атрибутами, которые либо принадлежат сущности, либо наследуются через связь;
* сущность обладает одним или несколькими атрибутами, которые однозначно идентифицируют каждый экземпляр сущности;
* каждая сущность может обладать любым количеством связей с другими сущностями модели.

Обращаясь к приведенным выше выдержкам из интервью, видно, что сущности, которые могут быть идентифицированы с главным менеджером - это автомашины и продавцы. Продавцу важны автомашины и связанные с их продажей данные. Для администратора важны покупатели, автомашины, продавцы и контракты. Исходя из этого, выделяются 4 сущности (автомашина, продавец, покупатель, контракт), которые изображаются на диаграмме следующим образом (рисунок 2).

http://gendocs.ru/docs/1/395/conv_1/file1_html_f47599c.png  
  
*Рис. 2.*   
  
Следующим шагом моделирования является идентификация связей.

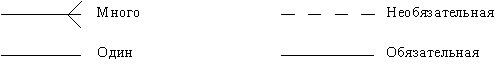
**Связь (Relationship)**- поименованная ассоциация между двумя сущностями, значимая для рассматриваемой предметной области. Связь - это ассоциация между сущностями, при которой, как правило, каждый экземпляр одной сущности, называемой родительской сущностью, ассоциирован с произвольным (в том числе нулевым) количеством экземпляров второй сущности, называемой сущностью-потомком, а каждый экземпляр сущности-потомка ассоциирован в точности с одним экземпляром сущности-родителя. Таким образом, экземпляр сущности-потомка может существовать только при существовании сущности родителя.

Связи может даваться имя, выражаемое грамматическим оборотом глагола и помещаемое возле линии связи. Имя каждой связи между двумя данными сущностями должно быть уникальным, но имена связей в модели не обязаны быть уникальными. Имя связи всегда формируется с точки зрения родителя, так что предложение может быть образовано соединением имени сущности-родителя, имени связи, выражения степени и имени сущности-потомка.

Например, связь продавца с контрактом может быть выражена следующим образом:

* продавец может получить вознаграждение за 1 или более контрактов;
* контракт должен быть инициирован ровно одним продавцом.

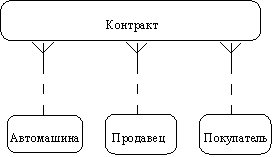
Степень связи и обязательность графически изображаются следующим образом (рисунок 3).

  
  
*Рис. 3.*

Таким образом, 2 предложения, описывающие связь продавца с контрактом, графически будут выражены следующим образом (рисунок 2.21).

http://gendocs.ru/docs/1/395/conv_1/file1_html_558e1683.png  
  
*Рис. 4.* 

Описав также связи остальных сущностей, получим следующую схему (рисунок 5).

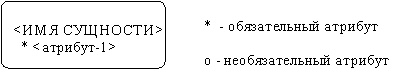
  
  
*Рис. 5.*

Последним шагом моделирования является идентификация атрибутов.

**Атрибут** - любая характеристика сущности, значимая для рассматриваемой предметной области и предназначенная для квалификации, идентификации, классификации, количественной характеристики или выражения состояния сущности. Атрибут представляет тип характеристик или свойств, ассоциированных со множеством реальных или абстрактных объектов (людей, мест, событий, состояний, идей, пар предметов и т.д.). Экземпляр атрибута - это определенная характеристика отдельного элемента множества. Экземпляр атрибута определяется типом характеристики и ее значением, называемым значением атрибута. В ER-модели атрибуты ассоциируются с конкретными сущностями. Таким образом, экземпляр сущности должен обладать единственным определенным значением для ассоциированного атрибута.

Атрибут может быть либо обязательным, либо необязательным (рисунок 2.23). Обязательность означает, что атрибут не может принимать неопределенных значений (null values). Атрибут может быть либо описательным (т.е. обычным дескриптором сущности), либо входить в состав уникального идентификатора (первичного ключа).

**Уникальный идентификатор** - это атрибут или совокупность атрибутов и/или связей, предназначенная для уникальной идентификации каждого экземпляра данного типа сущности. В случае полной идентификации каждый экземпляр данного типа сущности полностью идентифицируется своими собственными ключевыми атрибутами, в противном случае в его идентификации участвуют также атрибуты другой сущности-родителя (рисунок 6).

  
  
*Рис. 6.*   
  
  
  
*Рис. 7.*

Каждый атрибут идентифицируется уникальным именем, выражаемым грамматическим оборотом существительного, описывающим представляемую атрибутом характеристику. Атрибуты изображаются в виде списка имен внутри блока ассоциированной сущности, причем каждый атрибут занимает отдельную строку. Атрибуты, определяющие первичный ключ, размещаются наверху списка и выделяются знаком "#".

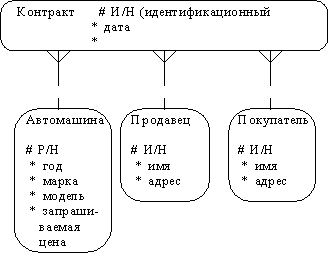
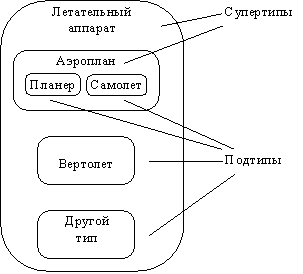
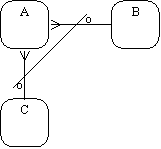
Каждая сущность должна обладать хотя бы одним возможным ключом. Возможный ключ сущности - это один или несколько атрибутов, чьи значения однозначно определяют каждый экземпляр сущности. При существовании нескольких возможных ключей один из них обозначается в качестве первичного ключа, а остальные - как альтернативные ключи.

С учетом имеющейся информации дополним построенную ранее диаграмму (рисунок 8).

Помимо перечисленных основных конструкций модель данных может содержать ряд дополнительных.

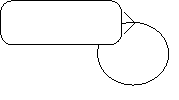
**Подтипы и супертипы:** одна сущность является обобщающим понятием для группы подобных сущностей (рисунок 9).

**Взаимно исключающие связи:** каждый экземпляр сущности участвует только в одной связи из группы взаимно исключающих связей (рисунок 10).

  
  
*Рис. 8.*   
  
  
  
*Рис. 9. Подтипы и супертипы*   
  
  
  
*Рис. 10. Взаимно исключающие связи*

**Рекурсивная связь:** сущность может быть связана сама с собой (рисунок 11). 

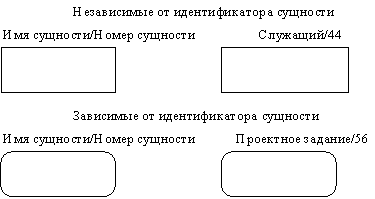
**Неперемещаемые (non-transferrable) связи:** экземпляр сущности не может быть перенесен из одного экземпляра связи в другой (рисунок 12).

  
  
*Рис. 11. Рекурсивная связь*   
  
http://gendocs.ru/docs/1/395/conv_1/file1_html_m125b283d.png  
  
*Рис. 12. Неперемещаемая связь* 

**2. Методология IDEF1**

Метод IDEF1, разработанный Т.Рэмей (T.Ramey), также основан на подходе П.Чена и позволяет построить модель данных, эквивалентную реляционной модели в третьей нормальной форме. В настоящее время на основе совершенствования методологии IDEF1 создана ее новая версия - методология IDEF1X. IDEF1X разработана с учетом таких требований, как простота изучения и возможность автоматизации. IDEF1X-диаграммы используются рядом распространенных CASE-средств (в частности, ERwin, Design/IDEF).

Сущность в методологии IDEF1X является независимой от идентификаторов или просто независимой, если каждый экземпляр сущности может быть однозначно идентифицирован без определения его отношений с другими сущностями. Сущность называется зависимой от идентификаторов или просто зависимой, если однозначная идентификация экземпляра сущности зависит от его отношения к другой сущности (рисунок 13).

  
  
*Рис. 13. Сущности*

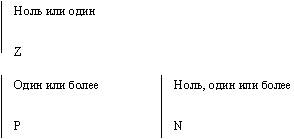
Каждой сущности присваивается уникальное имя и номер, разделяемые косой чертой "/" и помещаемые над блоком.

Связь может дополнительно определяться с помощью указания степени или мощности (количества экземпляров сущности-потомка, которое может существовать для каждого экземпляра сущности-родителя). В IDEF1X могут быть выражены следующие мощности связей:

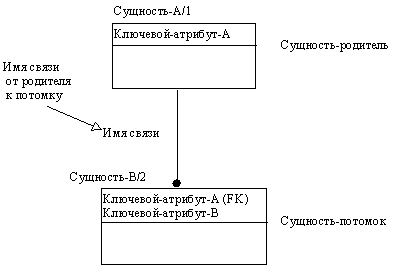
* каждый экземпляр сущности-родителя может иметь ноль, один или более связанных с ним экземпляров сущности-потомка;
* каждый экземпляр сущности-родителя должен иметь не менее одного связанного с ним экземпляра сущности-потомка;
* каждый экземпляр сущности-родителя должен иметь не более одного связанного с ним экземпляра сущности-потомка;
* каждый экземпляр сущности-родителя связан с некоторым фиксированным числом экземпляров сущности-потомка.

Если экземпляр сущности-потомка однозначно определяется своей связью с сущностью-родителем, то связь называется идентифицирующей, в противном случае - неидентифицирующей.

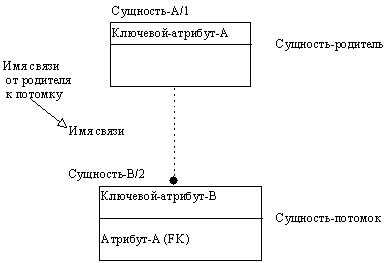
Связь изображается линией, проводимой между сущностью-родителем и сущностью-потомком с точкой на конце линии у сущности-потомка. Мощность связи обозначается как показано на рис. 14щность по умолчанию - N).

  
  
*Рис. 14. Мощность связи* 

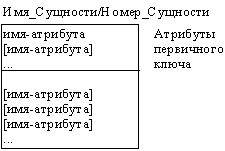
Идентифицирующая связь между сущностью-родителем и сущностью-потомком изображается сплошной линией (рисунок 15). Сущность-потомок в идентифицирующей связи является зависимой от идентификатора сущностью. Сущность-родитель в идентифицирующей связи может быть как независимой, так и зависимой от идентификатора сущностью (это определяется ее связями с другими сущностями).

  
  
*Рис.15. Идентифицирующая связь* 

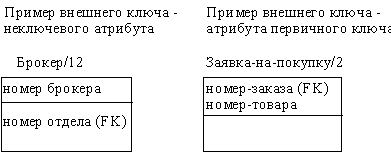
Пунктирная линия изображает неидентифицирующую связь (рисунок 16). Сущность-потомок в неидентифицирующей связи будет независимой от идентификатора, если она не является также сущностью-потомком в какой-либо идентифицирующей связи.

  
  
*Рис.17. Неидентифицирующая связь*

Атрибуты изображаются в виде списка имен внутри блока сущности. Атрибуты, определяющие первичный ключ, размещаются наверху списка и отделяются от других атрибутов горизонтальной чертой (рисунок 17).

  
  
*Рис. 18. Атрибуты и первичные ключи*

Сущности могут иметь также внешние ключи (Foreign Key), которые могут использоваться в качестве части или целого первичного ключа или неключевого атрибута. Внешний ключ изображается с помощью помещения внутрь блока сущности имен атрибутов, после которых следуют буквы FK в скобках (рисунок 19).

  
  
*Рис.19. Примеры внешних ключей*

**3. Подход, используемый в CASE-средстве Vantage Team Builder**

В CASE-средстве Vantage Team Builder (Westmount I-CASE) [14] используется один из вариантов нотации П. Чена. На ER-диаграммах сущность обозначается прямоугольником, содержащим имя сущности (рисунок 20), а связь - ромбом, связанным линией с каждой из взаимодействующих сущностей. Числа над линиями означают степень связи.

http://gendocs.ru/docs/1/395/conv_1/file1_html_299e527e.png  
  
*Рис. 20 Обозначение сущностей и связей*

Связи являются многонаправленными и могут иметь атрибуты (за исключением ключевых). Выделяют два вида связей:

* необязательная связь (optional);
* слабая связь (weak).

В **необязательной связи** (рисунок 21) могут участвовать не все экземпляры сущности.   
  
http://gendocs.ru/docs/1/395/conv_1/file1_html_60384a92.png  
  
*Рис. 21 Необязательная связь*

В отличие от необязательной связи в **полной (total)** связи участвуют все экземпляры хотя бы одной из сущностей. Это означает, что экземпляры такой связи существуют только при условии существования экземпляров другой сущности. Полная связь может иметь один из 4-х видов: обязательная связь, слабая связь, связь "супертип-подтип" и ассоциативная связь.

**Обязательная (mandatory) связь** описывает связь между "независимой" и "зависимой" сущностями. Все экземпляры зависимой ("обязательной") сущности могут существовать только при наличии экземпляров независимой ("необязательной") сущности, т.е. экземпляр "обязательной" сущности может существовать только при условии существования определенного экземпляра "необязательной" сущности.

В примере (рисунок 22) подразумевается, что каждый автомобиль имеет по крайней мере одного водителя, но не каждый служащий управляет машиной.

http://gendocs.ru/docs/1/395/conv_1/file1_html_m5b67f452.png  
  
*Рис. 22. Обязательная связь* 

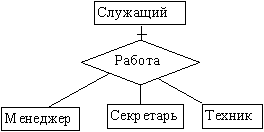
В **слабой связи** существование одной из сущностей, принадлежащей некоторому множеству ("слабой") зависит от существования определенной сущности, принадлежащей другому множеству ("сильной"), т.е. экземпляр "слабой" сущности может быть идентифицирован только посредством экземпляра "сильной" сущности. Ключ "сильной" сущности является частью составного ключа "слабой" сущности.

Слабая связь всегда является бинарной и подразумевает обязательную связь для "слабой" сущности. Сущность может быть "слабой" в одной связи и "сильной" в другой, но не может быть "слабой" более, чем в одной связи. Слабая связь может не иметь атрибутов.

Пример на рисунке 23: ключ (номер) строки в документе может не быть уникальным и должен быть дополнен ключом документа.

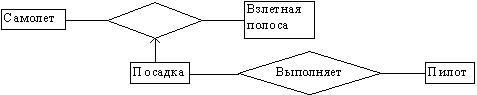
http://gendocs.ru/docs/1/395/conv_1/file1_html_m2584ca5f.png  
  
*Рис. 23 Слабая связь*

Связь "супертип-подтип" изображена на рисунке 24. Общие характеристики (атрибуты) типа определяются в сущности-супертипе, сущность-подтип наследует все характеристики супертипа. Экземпляр подтипа существует только при условии существования определенного экземпляра супертипа. Подтип не может иметь ключа (он импортирует ключ из супертипа). Сущность, являющаяся супертипом в одной связи, может быть подтипом в другой связи. Связь супертипа не может иметь атрибутов.

  
  
*Рис. 24вязь "супертип-подтип"*

В **ассоциативной связи** каждый экземпляр связи (ассоциативный объект) может существовать только при условии существования определенных экземпляров каждой из взаимосвязанных сущностей. Ассоциативный объект - объект, являющийся одновременно сущностью и связью. Ассоциативная связь - это связь между несколькими "независимыми" сущностями и одной "зависимой" сущностью. Связь между независимыми сущностями имеет атрибуты, которые определяются в зависимой сущности. Таким образом, зависимая сущность определяется в терминах атрибутов связи между остальными сущностями.

В примере на рисунке 25 самолет выполняет посадку на взлетную полосу в заданное время при определенной скорости и направлении ветра. Поскольку эти характеристики применимы только к конкретной посадке, они являются атрибутами посадки, а не самолета или взлетной полосы. Пилот, выполняющий посадку, связан гораздо сильнее с конкретной посадкой, чем с самолетом или взлетной полосой.

  
  
*Рис. 25 Ассоциативная связь*

Первичный ключ каждого типа сущности помечается звездочкой (\*).

ER-диаграмма должна подчиняться следующим правилам:

* каждая сущность, каждый атрибут и каждая связь должны иметь имя (связь супертипа или ассоциативная связь может не иметь имени);
* имя сущности должно быть уникально в рамках модели данных;
* имя атрибута должно быть уникально в рамках сущности;
* имя связи должно быть уникально, если для нее генерируется таблица БД;
* каждый атрибут должен иметь определение типа данных;
* сущность в необязательной связи должна иметь ключевой атрибут. То же самое относится к сильной сущности в слабой связи, супертипу в связи "супертип-подтип" и необязательной сущности в обязательной (полной) связи;
* подтип в свяи "супертип-подтип" не может иметь ключевой атрибут;
* в ассоциативной или слабой связи может быть только одна ассоциативная (слабая) сущность;
* связь не может быть одновременно обязательной, "супертип-подтип" или ассоциативной.

**4. Пример использования структурного подхода**

**4.1. Оисание предметной области**

В данном примере используется методология Yourdon [12], реализованная в CASE-средстве Vantage Team Builder.

В качестве предметной области используется описание работы видео библиотеки, которая получает запросы на фильмы от клиентов и ленты, возвращаемые клиентами. Запросы рассматриваются администрацией видео библиотеки с использованием информации о клиентах, фильмах и лентах. При этом проверяется и обновляется список арендованных лент, а также проверяются записи о членстве в библиотеке. Администрация контролирует также возвраты лент, используя информацию о фильмах, лентах и список арендованных лент, который обновляется. Обработка запросов на фильмы и возвратов лент включает следующие действия: если клиент не является членом библиотеки, он не имеет права на аренду. Если требуемый фильм имеется в наличии, администрация информирует клиента об арендной плате. Однако, если клиент просрочил срок возврата имеющихся у него лент, ему не разрешается брать новые фильмы. Когда лента возвращается, администрация рассчитывает арендную плату плюс пени за несвоевременный возврат.

Видеобиблиотека получает новые ленты от своих поставщиков. Когда новые ленты поступают в библиотеку, необходимая информация о них фиксируется. Информация о членстве в библиотеке содержится отдельно от записей об аренде лент.

Администрация библиотеки регулярно готовит отчеты за определенный период времени о членах библиотеки, поставщиках лент, выдаче определенных лент и лентах, приобретенных библиотекой.

**4.2. Организация проекта**

Весь проект разделяется на 4 фазы: анализ, глобальное проектирование (проектирование архитектуры системы), детальное проектирование и реализация (программирование).

На фазе анализа строится модель среды (Environmental Model). Построение модели среды включает:

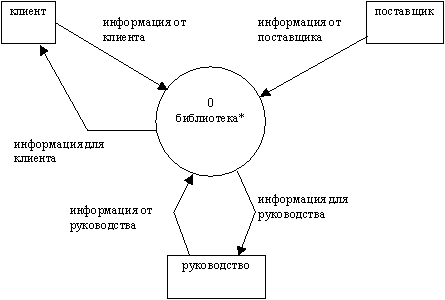
* анализ поведения системы (определение назначения ИС, построение начальной контекстной диаграммы потоков данных (DFD) и формирование матрицы списка событий (ELM), построение контекстных диаграмм);
* анализ данных (определение состава потоков данных и построение диаграмм структур данных (DSD), конструирование глобальной модели данных в виде ER-диаграммы).

Назначение ИС определяет соглашение между проектировщиками и заказчиками относительно назначения будущей ИС, общее описание ИС для самих проектировщиков и границы ИС. Назначение фиксируется как текстовый комментарий в "нулевом" процессе контекстной диаграммы.

Например, в данном случае назначение ИС формулируется следующим образом: ведение базы данных о членах библиотеки, фильмах, аренде и поставщиках. При этом руководство библиотеки должно иметь возможность получать различные виды отчетов для выполнения своих задач.

Перед построением контекстной DFD необходимо проанализировать внешние события (внешние объекты), оказывающие влияние на функционирование библиотеки. Эти объекты взаимодействуют с ИС путем информационного обмена с ней.

Из описания предметной области следует, что в процессе работы библиотеки участвуют следующие группы людей: клиенты, поставщики и руководство. Эти группы являются внешними объектами. Они не только взаимодействуют с системой, но также определяют ее границы и изображаются на начальной контекстной DFD как терминаторы (внешние сущности).

Начальная контекстная диаграмма изображена на рисунке 26. В отличие от нотации Gane/Sarson внешние сущности обозначаются обычными прямоугольниками, а процессы - окружностями.   
  
*Рис. 26. Начальная контекстная диаграмма*

Список событий строится в виде матрицы (ELM) и описывает различные действия внешних сущностей и реакцию ИС на них. Эти действия представляют собой внешние события, воздействующие на библиотеку. Различают следующие типы событий:

|  |  |
| --- | --- |
| **Аббревиатура** | **Тип** |
| NC | Нормальное управление |
| ND | Нормальные данные |
| NCD | Нормальное управление/данные |
| TC | Временное управление |
| TD | Временные данные |
| TCD | Временное управление/данные |

Все действия помечаются как нормальные данные. Эти данные являются событиями, которые ИС воспринимает непосредственно, например, изменение адреса клиента, которое должно быть сразу зарегистрировано. Они появляются в DFD в качестве содержимого потоков данных.

Матрица списка событий имеет следующий вид:

|  |  |  |  |
| --- | --- | --- | --- |
| № | **Описание** | **Тип** | **Реакция** |
| **1** | Клиент желает стать членом библиотеки | ND | Регистрация клиента в качестве члена библиотеки |
| **2** | Клиент сообщает об изменении адреса | ND | Регистрация измененного адреса клиента |
| **3** | Клиент запрашивает аренду фильма | ND | Рассмотрение запроса |
| **4** | Клиент возвращает фильм | ND | Регистрация возврата |
| **5** | Руководство предоставляет полномочия новому поставщику | ND | Регистрация поставщика |
| **6** | Поставщик сообщает об изменении адреса | ND | Регистрация измененного адреса поставщика |
| **7** | Поставщик направляет фильм в библиотеку | ND | Получение нового фильма |
| **8** | Руководство запрашивает новый отчет | ND | Формирование требуемого отчета для руководства |

Для завершения анализа функционального аспекта поведения системы строится полная контекстная диаграмма, включающая диаграмму нулевого уровня. При этом процесс "библиотека" декомпозируется на 4 процесса, отражающие основные виды административной деятельности библиотеки. Существующие "абстрактные" потоки данных между терминаторами и процессами трансформируются в потоки, представляющие обмен данными на более конкретном уровне. Список событий показывает, какие потоки существуют на этом уровне: каждое событие из списка должно формировать некоторый поток (событие формирует входной поток, реакция - выходной поток). Один "абстрактный" поток может быть разделен на более чем один "конкретный" поток.

|  |  |
| --- | --- |
| **Потоки на диаграмме верхнего уровня** | **Потоки на диаграмме нулевого уровня** |
| Информация от клиента | Данные о клиенте, Запрос об аренде |
| Информация для клиента | Членская карточка, Ответ на запрос об аренде |
| Информация от руководства | Запрос отчета о новых членах, Новый поставщик, Запрос отчета о поставщиках, Запрос отчета об аренде, Запрос отчета о фильмах |
| Информация для руководства | Отчет о новых членах, Отчет о поставщиках, Отчет об аренде, Отчет о фильмах |
| Информация от поставщика | Данные о поставщике, Новые фильмы |

На приведенной DFD (рисунок 27) накопитель данных "библиотека" является глобальным или абстрактным представлением хранилища данных.

Анализ функционального аспекта поведения системы дает представление об обмене и преобразовании данных в системе. Взаимосвязь между "абстрактными" потоками данных и "конкретными" потоками данных на диаграмме нулевого уровня выражается в диаграммах структур данных (рисунок 28).

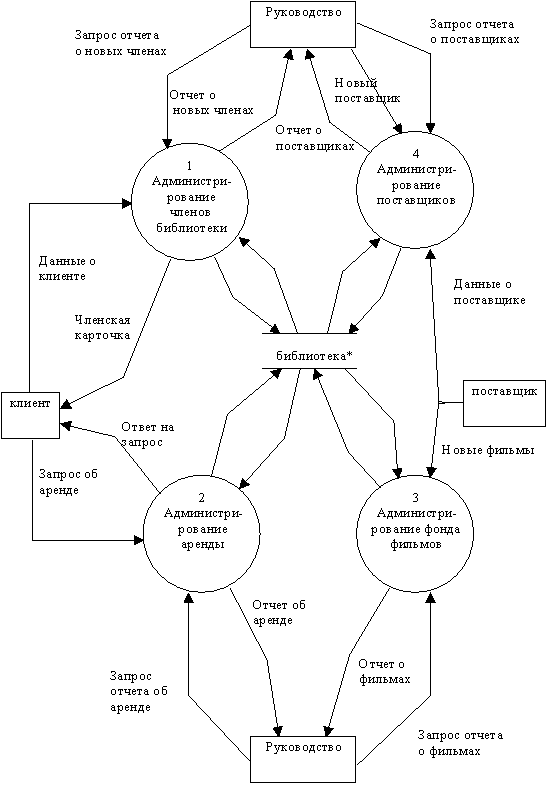
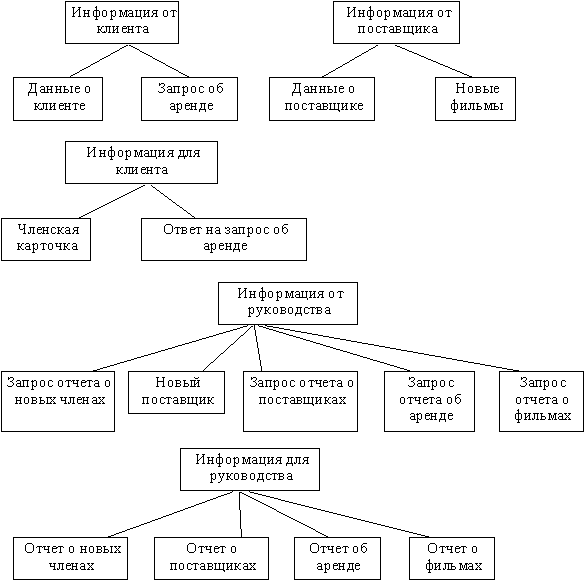
На фазе анализа строится глобальная модель данных, представляемая в виде диаграммы "сущность-связь" (рисунок 29).

Между различными типами диаграмм существуют следующие взаимосвязи:

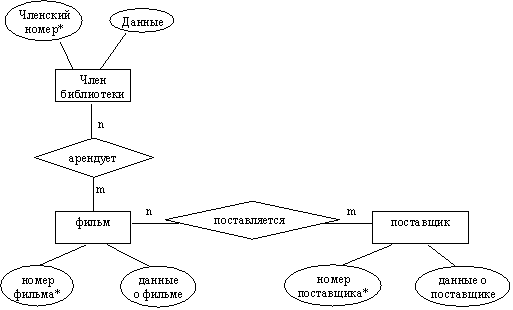
* ELM-DFD: события - входные потоки, реакции - выходные потоки
* DFD-DSD: потоки данных - структуры данных верхнего уровня
* DFD-ERD: накопители данных - ER-диаграммы
* DSD-ERD: структуры данных нижнего уровня - атрибуты сущностей

На фазе проектирования архитектуры строится предметная модель. Процесс построения предметной модели включает в себя:

* детальное описание функционирования системы;
* дальнейший анализ используемых данных и построение логической модели данных для последующего проектирования базы данных;
* определение структуры пользовательского интерфейса, спецификации форм и порядка их появления;
* уточнение диаграмм потоков данных и списка событий, выделение среди процессов нижнего уровня интерактивных и неинтерактивных, определение для них миниспецификаций.

  
  
*Рис. 27. Контекстная диаграмма*  
  
  
  
*Рис. 28. Диаграмма структур данных*   
  
Результатами проектирования архитектуры являются: 

* модель процессов (диаграммы архитектуры системы (SAD) и миниспецификации на структурированном языке);
* модель данных (ERD и подсхемы ERD);
* модель пользовательского интерфейса (классификация процессов на интерактивные и неинтерактивные функции, диаграмма последовательности форм (FSD - Form Sequence Diagram), показывающая, какие формы появляются в приложении и в каком порядке. На FSD фиксируется набор и структура вызовов экранных форм. Диаграммы FSD образуют иерархию, на вершине которой находится главная форма приложения, реализующего подсистему. На втором уровне находятся формы, реализующие процессы нижнего уровня функциональной структуры, зафиксированной на диаграммах SAD.

  
  
*Рис. 29. Диаграмма "сущность-связь"* 

На фазе детального проектирования строится модульная модель. Под модульной моделью понимается реальная модель проектируемой прикладной системы. Процесс ее построения включает в себя:

* уточнение модели базы данных для последующей генерации SQL-предложений;
* уточнение структуры пользовательского интерфейса;
* построение структурных схем, отражающих логику работы пользовательского интерфейса и модель бизнес-логики (Structure Charts Diagram - SCD) и привязка их к формам.

Результатами детального проектирования являются: 

* модель процессов (структурные схемы интерактивных и неинтерактивных функций);
* модель данных (определение в ERD всех необходимых параметров для приложений);
* модель пользовательского интерфейса (диаграмма последовательности форм (FSD), показывающая, какие формы появляются в приложении и в каком порядке, взаимосвязь между каждой формой и определенной структурной схемой, взаимосвязь между каждой формой и одной или более сущностями в ERD).

На фазе реализации строится реализационная модель. Процесс ее построения включает в себя:

* генерацию SQL-предложений, определяющих структуру целевой БД (таблицы, индексы, ограничения целостности);
* уточнение структурных схем (SCD) и диаграмм последовательности форм (FSD) с последующей генерацией кода приложений.

На основе анализа потоков данных и взаимодействия процессов с хранилищами данных осуществляется окончательное выделение подсистем (предварительное должно было быть сделано и зафиксировано на этапе формулировки требований в техническом задании). При выделении подсистем необходимо руководствоваться принципом функциональной связанности и принципом минимизации информационной зависимости. Необходимо учитывать, что на основании таких элементов подсистемы как процессы и данные на этапе разработки должно быть создано приложение, способное функционировать самостоятельно. С другой стороны при группировке процессов и данных в подсистемы необходимо учитывать требования к конфигурированию продукта, если они были сформулированы на этапе анализа.

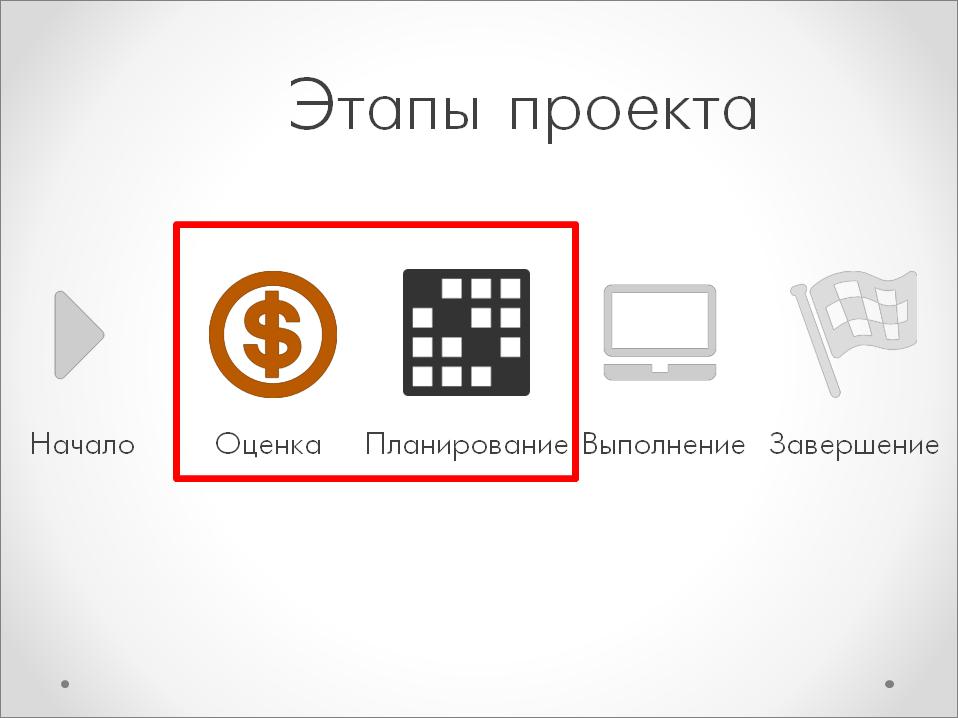
**Лекция 2. Планирование и оценка проекта**

Здесь приводится очень упрощенная схема проекта, всего 5 этапов.

Этап «**Начало проекта**» - здесь мы говорим про некую инициацию проекта, когда появляется заказчик, который говорит нам: «Я хочу чудо».

* После того, как заказчик это чудо захотел, он хочет понять, сколько это чудо стоит. Соответственно, мы переходим на этап «**Оценка проекта**».
* Потом на этапе «**Планирование проекта**» у нас появляются какие-то сроки.
* Дальше - этап «**Выполнение**» проекта.
* И потом - этап «**Завершение проекта**».

Понятно, что это очень условное деление проекта на этапы, потому что я пропустил этапы согласования, подписания договоров и т.д.



Сегодня мы будем обсуждать два первых этапа – этапы «Оценка проекта» и этап «Планирование проекта». Я их специально выделил на слайде.

## Этап оценки проекта

Итак, как мы делаем оценку проекта?

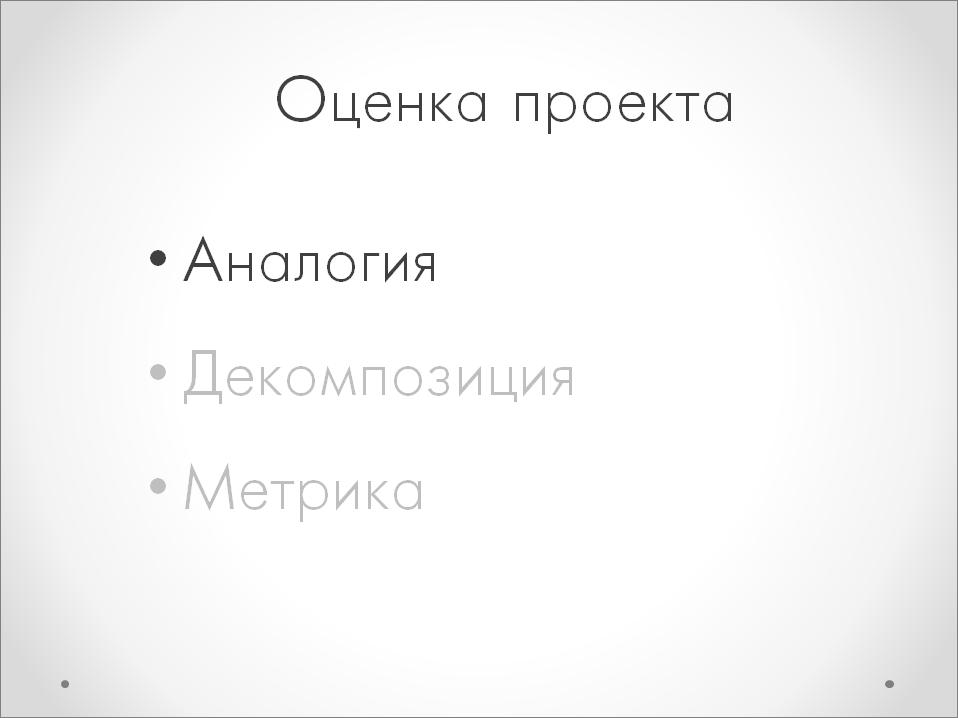


Я хотел бы начать с цитаты Уинстона Черчилля, который как-то сказал: «Любой умный человек может составить план победы в войне, если он не отвечает за выполнение этого плана».

И здесь мы приходим к осознанию глобальной проблемы, которая часто возникает при оценке. **Оценок бывает**, как правило, **две**:

* Первая оценка: приходит заказчик и спрашивает, сколько это будет стоить.  
  Мы ему говорим – «пять логанов». Заказчик говорит – «нет, дорого».  
  Тогда мы ему говорим – «ну, тогда два». Заказчик говорит – «два? Нормально».  
  Это – **продажная оценка**.
* И вторая оценка – это уже более-менее **реальная оценка**. Когда мы начинаем двигаться дальше к исполнению этого проекта и понимаем, что все-таки два логана – это было, наверное, маловато, что все-таки проект стоит не меньше четырех логанов.

И вот это **совмещение продажной** **и реальной оценки**, по которой потом будут люди делать, это и есть самая **глобальная проблема**, которая часто возникает. Потому что проекты продают одни люди, а оценивают – другие.



Исходя из своего опыта, я выделяю **три способа проведения оценки**:

* Аналогия
* Декомпозиция
* Метрика

Давайте каждый из способов отдельно обсудим.

**Первый способ оценки – аналогия.**

Вроде все понятно: у нас был какой-то проект, который мы делали два года назад. К нам приходит заказчик и спрашивает: «Сколько это будет стоить?»

Мы с этим вопросом идем к нашему человеку, который делал этот предыдущий проект. Он чешет голову, вспоминает и говорит: мы делали этот проект за два логана, потратили на него 2000 часов.

Плюсы такой оценки понятны – это быстрая оценка.

Какие могут возникнуть проблемы, если вы будете оценивать свои проекты таким способом?

* Проблема номер 1. Нужно обоснование, в любом случае нужно будет написать, почему это было именно так.
* Другая проблема – нет никакой гарантии того, что в предыдущий раз мы сделали этот проект правильно. Даже не того, что мы сделали этот проект неправильно, просто мы действительно могли бы сделать этот проект дешевле. Аналогия позволяет оценить только какую-то верхнюю границу примерных трудозатрат.
* Еще проблема состоит в том, что с помощью аналогии можно оценивать, как правило, только маленькие или средние проекты. Какие-то крупные проекты по аналогии оценить невозможно, потому что слишком много возникает нюансов и тонкостей.

Аналогия – это интуитивно понятный процесс, здесь ничего сложного нет.

**Следующий способ оценки – декомпозиция**

Собственно, декомпозиция является стандартом де-факто всех оценок. 90% всех оценок делаются с помощью декомпозиции.

Декомпозиция – это когда мы берем некий список задач и декомпозируем их на более мелкие подзадачи. Начинаем их делить и, собственно, переходим уже к более мелким обозримым задачам, которые мы можем оценить.

Плюсы декомпозиции понятны. С помощью этого способа мы можем оценить самые большие проекты. Если у нас есть какой-то огромный проект, то мы можем его в принципе декомпозировать до более мелких частей, реально обозримых, которые мы можем реально оценить.

Какие у нас здесь возникают ключевые проблемы?

* Когда мы делим проект на задачи, то может потеряться интеграционная составляющая по взаимодействию этих частей между собой. И если у вас нет человека, который мог бы в голове представить этот проект, определить взаимосвязь выделенных блоков между собой, тогда вам нужно будет делать где-нибудь внизу приписку: «Интеграционная составляющая проекта – 25%»(25% – это в лучшем случае, иногда 50%). Ведь если мы не можем правильно оценить интеграционные затраты, то, соответственно, у нас возникают дополнительные риски, которые нужно закрывать. Я думаю, что на интеграционный момент всегда нужно закладывать не меньше 10-25%.
* У декомпозиции есть еще одна проблема – это трудоемкость ее проведения. Есть более-менее простые проекты, которые достаточно быстро можно оценить, декомпозировать. Но встречаются настолько сложные проекты, что только сам процесс их оценки будет составлять 40-50-80 и более часов. Это уже достаточно серьезная работа, которая требует выделения отдельных ресурсов.

Итак, основная мысль следующая – как бы мы ни делили проект на подзадачи, всегда должен быть некий интегрирующий человек, который представляет себе весь проект в целом. Здесь это однозначно.

**Третий способ оценки - метрика**

Метрика – один из моих любимых способов оценки. Этот способ мне кажется самым интересным, потому что он позволяет с высокой точностью и минимальными трудозатратами оценивать достаточно сложные проекты.

Что такое метрика как способ оценки? Мы выделяем некое количественное выражение проекта и по этой метрике оцениваем отдельную единицу трудозатрат на отдельную единицу этой метрики. Соответственно, дальше мы получаем эту количественную оценку и просто умножаем трудозатраты одной единицы на общее количество метрики. В результате получаем оценку трудозатрат по всему проекту.

Очень хорошо поддаются метрике проекты перевода конфигураций с обычных форм на управляемые. Примерная средняя оценка перевода объекта с обычной формы на управляемую – три часа. То есть, мы просто считаем количество нетиповых объектов и умножаем их количество на 3 часа – получаем общую трудоемкость. Такая оценка мне очень нравится, потому что она позволяет быстро и достаточно точно оценить проект.

Какая проблема здесь есть?

* Самая главная проблема в том, что далеко не все проекты поддаются метрике.

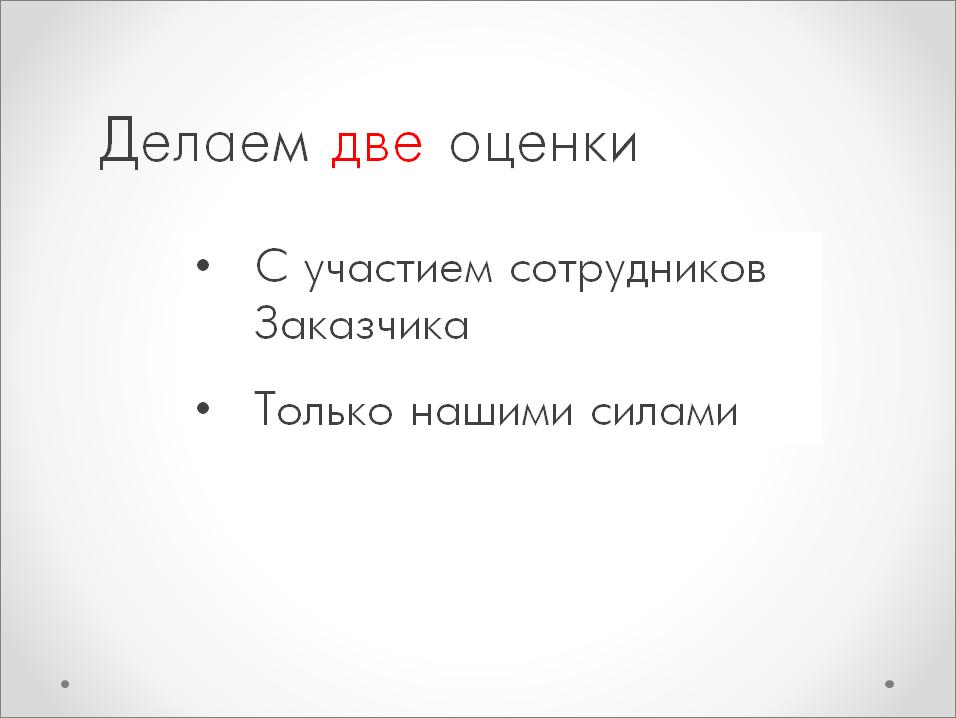
Если у меня получается, я всегда стараюсь оценивать планируемую трудоемкость проекта по метрике. Потому что в этом случае никакого дополнительного обоснования уже не требуется, метрика уже достаточно четко показывает заказчику, как мы получаем итоговую оценку.

Теперь такой вопрос – **как мы делаем оценку** **на практике**?

Мы**должны делать** не одну, а **две оценки**.

* Мы должны сделать оценку **с участием сотрудников заказчика**

И мы должны сделать оценку **только нашими силами**.



Что это означает?



Здесь указана декомпозиция по задачам. Есть первая задача и вторая. У нас есть две колонки с трудозатратами – колонка №1 и колонка №2.

* В колонке №1 мы указываем трудоемкость этой задачи с минимальным нашим привлечением,
* В колонке №2 мы указываем общую трудоемкость этой задачи.

В колонке №1 может быть указан ноль. Это означает, что заказчик может справиться с этой задачей полностью самостоятельно, без нашего привлечения, если выделит от себя некоего специалиста, некий ресурс, который эту задачу выполнит.

А дальше мы оцениваем те задачи, которые требуют нашего обязательного участия, которые можем выполнить только мы. Например, мы считаем, что разработать правила обмена и перенести остатки по счетам можем только мы, а другие задачи можем выполнить и мы, и заказчик. У нас очень часто на встречах заказчик говорит – «это мы делаем сами, а это делаете вы». Такой способ оценки очень четко позволяет заказчику определить для себя общий объем затрат по этому проекту и дальше уже пройтись по нему.

Точная такая же схема для внутренних заказчиков. На внутренних проектах тоже есть задачи, которые должны выполнять сотрудники других отделов, например, приходится привлекать специалистов на тестирование и т. д. Это постоянно бывает, потому что, как правило, проекты не делаются одним человеком без привлечения каких-то других специалистов.

**Двойная оценка трудоемкости** – это один из самых удачных примеров конкретизации объема планируемых затрат.

Еще один ключевой момент – **мы договариваемся о том, чего мы не делаем**. И это мы стараемся прописывать достаточно четко.

* То, что мы делаем, вроде и так понятно. У нас есть некий перечень того, что мы делаем, это указано в наших задачах, в нашем перечне, в нашей оценке.
* Но **в любой оценке** мы всегда **приводим ограничение допущения** – либо в сопроводительном письме, либо в самой оценке, либо прямо в задаче расписываем комментарии о том, что мы не делаем в ней. И эти ограничения, допущения и договоренности о том, чего мы не делаем, являются очень действенным способом снизить наши риски.

Мы стараемся всегда это прописывать. **Этот способ позволяет** наиболее эффективно **привести продажную оценку к реальной**, сблизить их между собой. Заказчик начинает понимать, что за эту цену он получит вот такой объем работ, ограниченный вот с этой стороны, с этой стороны и с этой стороны. Причем эти ограничения и допущения мы всегда **включаем в договор**, чтобы заказчик понимал, что объем проекта будет именно таким.

По этапу оценки проекта я решил сам себе задать вопросы, которые могут родиться у вас в ходе чтения статьи.

* Первый вопрос – **как понять адекватность оценки**?
  + Адекватность понять можно в первую очередь следующим образом – **нужно в эту оценку поверить**. Адекватность – это когда вы начинаете понимать, что проект будет вот таким, и он будет происходить вот в такой последовательности, и на вот такой объем будут требоваться вот такие трудозатраты.
  + Второй способ проверки адекватности – это **попросить** кого-нибудь **сделать повторную оценку** той же работы. Примерно треть оценок, которые нам приходится выполнять, связаны с тем, что наши заказчики просят нас оценить какой-то проект. Мы понимаем, что это повторная оценка (для того, чтобы с кем-то сравниться), но мы все равно ее делаем для того, чтобы удовлетворить потребности клиента.  
    Здесь сразу возникает следующий вопрос – если у вас была одна оценка (где было два логана) и появилась другая (где десять логанов), то, как понять, какая из этих оценок более адекватна?  
    По большому счету, единственный способ найти ответ на этот вопрос – это понять, что написано в этой оценке и просто в те цифры, которые вам предъявляют. Если вы не верите, значит, эта оценка неадекватна для вас. Других способов оценки адекватности нет.
  + Второй вопрос – **как подрядчик включает в оценку свои риски**?  
    Очень просто. Он просто пишет **в договоре отдельным пунктом** риски проекта 10% или 25%. Конечно, очень часто бывает ситуация, когда такой отдельной строчки в договоре нет, и просто все трудозатраты по всем задачам умножаются на 0.25. Но мы так стараемся не делать. Мы всегда отдельно обговариваем, что есть определенные риски, которые мы не можем смоделировать.
  + Третий вопрос – **когда возникают риски**?  
    Риски возникают на разных проектах по разным причинам. Они могут возникать **на согласовании, на трудоемкости ввода**, на других этапах. Естественно, все эти риски мы стараемся каким-то образом озвучить, зафиксировать, что здесь мы видим такие риски, и мы в них закладываемся. Это – некий НЗ, который мы для себя фиксируем «на всякий случай», стараясь в этот НЗ не залезать. И обосновываем для заказчика, что если мы в этот НЗ залезаем, то это происходит по вот таким причинам.
  + И последний вопрос – **что самое важное в оценке**?  
    Я об этом уже сказал – **в нее нужно поверить**. Других вариантов просто нет. Если вы верите в эту оценку – это нормальная оценка, правильная. Если не верите, значит, что-то в ней не так.

## Этап планирования проекта

Переходим к следующему этапу – этап планирования. Как мы планируем проекты?

Здесь я хотел бы привести цитату Дуэйта Эйзенхауэра: «Когда готовишься к сражению, планировать бессмысленно, но планирование необходимо».

Мне кажется, эта цитата достаточно четко демонстрирует тот факт, что без планирования ни один проект не обходится. При этом я не помню, чтобы хоть один проект проходил по тому запланированному перечню работ, в те сроки, которые были заложены на этапе первоначального планирования.

Какие вообще **варианты планирования** проекта у нас есть? Их тоже три.

* Первый вариант планирования – это **планирование от старта проекта**. Когда у нас есть перечень работ, указанный в оценке, и мы просто **раскладываем эти работы на календарный график**. А дальше уже от этого откладываем какие-то сроки выполнения каждого из этапов. Это разумный, часто встречающийся подход.
* Следующий вариант планирования – **это планирование от даты X**. Здесь тот же самый принцип, что и в первом варианте, только здесь мы планируем от конца, когда у нас есть некая конечная дата – например, дата запуска проекта (дата старта новой базы) – 1 января 2014 года. Соответственно, мы **раскладываем перечень наших работ от этой даты назад**.

И последний вариант планирования – **планирование на короткий период**. Такое планирование относится больше к технологиям Agile, SCRUM, когда у нас планирование идет не на весь проект, а на какой-то короткий обозримый промежуток в течение одной-двух недель. По идее, промежутки планирования при этих технологиях не превышают двух недель. Соответственно, у нас есть некий план на короткий промежуток, мы его выполняем и двигаемся дальше. Планирование на короткий промежуток тоже является одним из способов планирования.

**Как не надо планировать**?

Очень часто на старте проекта заказчики и исполнители пытаются спланировать проект следующим образом:

* **Все сразу**
* **Детально**
* **И более чем на 6 месяцев**.

Не работает. Я не помню такого варианта, чтобы проект был спланирован сразу детально на срок более чем 6 месяцев, и это сработало.

Хорошо работает, когда у нас есть какие-то хорошо спланированные детали. Или может быть хорошо спланирован первый этап длительностью 1-2 месяца, а затем для следующих этапов просто указаны ориентировочные даты о том, что задача будет выполняться с такого-то по такое-то. Я считаю, что такой план может сработать с большей вероятностью.

Зачем пытаться спланировать сразу все? Сроки и так более-менее понятны, мы их планируем изначально по большим этапам.

**Что надо учитывать при планировании?**

* При планировании нужно обязательно учитывать интенсивность обучения. **Разные люди обучаются по-разному** – кому-то требуется 2 дня на то, чтобы обучиться, а кому-то требуется 2 недели. Это достаточно существенная разница. Поэтому, когда мы приходим к этапу обучения, внедрения, когда уже идет запуск в опытную эксплуатацию – разная скорость обучения может сыграть свою роль. Это надо учитывать и понимать, что группы обучения очень разнородны.
* **Тестирование**. На этапе перевода в опытную, промышленную эксплуатацию тестированием занимается заказчик. И при планировании нужно закладываться на то, что этот этап может быть очень длительным, так как тестирование**обычно выполняется пользователями заказчика в свободное от работы время**. А поскольку у них 100% рабочего времени занимает выполнение их первоочередных рабочих обязанностей, то этап тестирования, как правило, затягивается надолго. И если по вашим оценкам трудозатраты на тестирование должны составить 8 часов, можно спокойно увеличивать их в 4 раза.  
  Я сейчас говорю не про тестирование со стороны разработчика, потому что в любом случае работу принимает заказчик. И вот эта **приемка работ** – она, как правило, **может занимать очень длительное время**. На это нужно закладываться при планировании.
* Тестирование и **загрузка ресурсов Заказчика** – это связанные между собой вещи, то есть очень часто **пользователи заказчика заняты** **так, что на проект у них времени не остается**, поэтому одни и те же трудозатраты раскладываются на гораздо более длительный срок. Растягиваются и могут растягиваться очень сильно. Это тоже нужно учитывать и при планировании закладывать на это дополнительное время (по длительности задачи).
* **Объемы обрабатываемых данных**. Здесь речь идет о том, что базы могут быть очень большими. Например, необходимо понимать, что перенос данных может проходить со сбоями. Поэтому если мы закладываем на него 40 часов, то на самом деле **из-за большого объема данных** эта длительность может быть гораздо больше просто из-за того, что **могут быть сбои, перезагрузки** и т.д. На эти все вещи нужно закладываться минимум раза в два.

**Время на согласование документов**.

Я бы сказал, что этот пункт – просто бич. Времени на согласование может потребоваться просто пропасть. Поэтому нужно прямо выделять вехи для согласования тех или иных документов. **Документы могут согласовываться неделями**. Часто происходит так, что сам документ мы сделали за неделю, и потом еще три недели мы его согласовываем.  
Время на согласование документов нужно закладывать очень серьезное, очень большое. **Особенно в крупных организациях**.

По этапу планирования проекта я тоже набросал очевидный список вопросов:

* **Нужно ли планировать ресурсы с самого начала**?Крупными мазками, наверное, надо. В любом случае надо понимать, кто будет делать задачу – разработчик или консультант. Но **указывать** при этом **конкретные персоналии** мне кажется, **не стоит**.  
  Когда мы планируем на период больше, чем 2 недели, поименно указывать, кто будет выполнять ту или иную задачу – бессмысленно. Через 2 недели смысла в вашем планировании особого не будет. Человек может заболеть, может пропасть, может уйти в отпуск. Но мы можем заложить конкретные ресурсы на проект в целом, просто чтобы на каком-то длительном этапе понимать, что этот человек нам может понадобиться. При этом мы не говорим, что он нам понадобится на этой конкретной задаче. Он просто проходит у нас, как некий ресурс на этом проекте. А на какой конкретно задаче мы его используем, мы в рамках долгосрочного планирования закладывать не должны. Вообще реальное распределение ресурсов по задачам должно происходить в ходе реализации, диспетчеризации задач в рамках одно-двух недельных этапов планирования. Планировать сразу, говорить, что через 3 месяца Вася Иванов будет делать такую-то задачу – это «вилами по воде писано» или «пальцем в потолок».
* Например, у меня была ситуация – на проекте был определенный круг бизнес-пользователей заказчика, которым я должен быть демонстрировать свою работу. Соответственно, когда я раскидал задачи, связанные с ними, по месяцу, я не зарезервировал их время на работу на нашем проекте. И в результате получилось, что я пришел на первое согласование, а половины людей нет: у одних – отпуска, у других – другие задачи и т.д. Пришлось перепланировать.
* В итоге получается, что у нас, технических руководителей проектов, есть **задача** в целом, есть ресурсы на проекте и наша цель – **сохранить эту команду проекта на все время подготовки проекта**. Все. По большому счету, не важно, куда человек с проекта может уйти – в отпуск или не в отпуск, на другой проект или куда-то еще. Поэтому руководителю проекта приходится, в том числе, планировать отпуска тех сотрудников, которые входят в эту команду, и для этого мы обычно просто договариваемся с человеком: пока идет проект, в отпуска не уходим – конечно, если у нас проект обозримой длительности по времени (три-пять-шесть месяцев).
* **Какой этап самый длительный**?
* Непростой вопрос. В разных проектах по-разному. Я считаю, что самый длительный, самый сложный этап – это **этап внедрения**, когда мы что-то накодировали и теперь начинаем это все внедрять – переносить остатки, обучать пользователей. Вот этот этап перехода от некоего концепта к опытной промышленной эксплуатации , пожалуй, занимает большую часть времени, чем другие этапы проекта.
* И последний вопрос – **можно ли не планировать**?  
  Можно. Но не нужно. Если перефразировать Эйзенхауэра – ни одна битва в истории человечества не прошла так, как ее задумывал полководец. Но без планирования победы не бывает.

**Планирование необходимо**. В любом случае мы должны планировать проект в целом по этапам, по задачам. И что гораздо правильнее - именно контролировать ход исполнения проекта, контролировать попадание в разработанный план и работать с изменениями. Но это уже относится не к этапам оценки и планирования, а больше к этапу исполнения проекта.

**Заключение**

Очень **часто проекты находятся в состоянии «перманентного планирования**». Но надо уже когда-то начинать воевать. Потому что там что-то планируют, потом перепланируют, создают первую оценку, потом – вторую, третью, потом опять что-то планируют…

То время и ресурсы, которые были потрачены на оценку и планирование, гораздо более эффективно можно было бы потратить на то, чтобы начать что-то делать. Потому что вот эта попытка охватить все сразу, детально и более чем на 6 месяцев, не очень хорошо работает. **Гораздо правильнее уже начать делать**.