Лекция 1. Введение. Базовые понятия.

Аннотация.

Вводная информация: цели курса, его состав, требования к студентам, система оценок и прочая орг. информация.

Информация и ее виды. Меры информации, критерии качества.

Оглавление

Введение: организация занятий, цели курса, виды контрольных мероприятий.

Информация и ее виды.

[Лидовский]

Кибернетика, теория информации.

Теория информации рассматривается как существенная часть кибернетики.

Кибернетика – это наука об общих законах получения, хранения, передачи и переработки информации.

Объектом исследования кибернетики являются все управляемые системы.

Примеры кибернетических систем: автоматические регуляторы в технике, ЭВМ, мозг человека или животных, биологическая популяция, социум. Часто кибернетику связывают с методами искусственного интеллекта, т.к. она разрабатывает принципы создания систем управления и систем автоматизации умственного труда.

Основными разделами кибернетики считаются: теория информации, теория алгоритмов, теория автоматов, исследование операций, теория оптимального управления и теория распознавания образов.

Родоначальниками кибернетики считаются американские ученые Норберт Винер и Клод Шеннон. Клод Шеннон – основоположник теории информации.

Но́рберт Ви́нер (англ. Norbert Wiener; 26 ноября 1894, Колумбия, штат Миссури, США — 18 марта 1964, Стокгольм, Швеция) — американский учёный, выдающийся математик и философ, основоположник кибернетики и теории искусственного интеллекта.

В 4 года Винер уже был допущен к родительской библиотеке, а в 7 лет написал свой первый научный трактат по дарвинизму. Норберт никогда по-настоящему не учился в средней школе. Зато 11 лет от роду он поступил в престижный Тафт-колледж, который закончил с отличием уже через три года получив степень бакалавра искусств.

В 18 лет Норберт Винер получил степени доктора философии по математической логике в Корнельском и Гарвардском университетах. В девятнадцатилетнем возрасте доктор Винер был приглашён на кафедру математики Массачусетского технологического института.

Винер ввел основную категорию кибернетики – управление, показал существенные отличия этой категории от других, например, энергии, описал несколько задач, типичных для кибернетики и привлек всеобщее внимание к особой роли вычислительных машин, считая их индикатором наступления новой НТР. Выделение категории управления позволило Винеру воспользоваться понятием информации, положив в основу кибернетики изучение законов передачи и преобразования информации.

Кибернетика — наука об оптимальном управлении сложными динамическими системами, изучающая общие принципы управления и связи, лежащие в основе работы самых разнообразных по природе систем — от самонаводящих ракет-снарядов и быстродействующих вычислительных машин до сложного живого организма.

Оптимальное управление — это перевод системы в новое состояние с выполнением некоторого критерия оптимальности, например, минимизации затрат времени, труда, веществ или энергии.

Сущность принципа управления заключается в том, что движение и действие больших масс или передача и преобразование больших количеств энергии направляется и контролируется при помощи небольших количеств энергии, несущих информацию. Этот принцип управления лежит в основе организации и действия любых управляемых систем.

Теория информации тесно связана с такими разделами математики, как теория вероятностей и математическая статистика, которые представляют для нее математический фундамент. Теория информации представляет собой математическую теорию, посвященную измерению информации, ее потока, характеристик канала связи и т.п., особенно применительно к радио, телевидению и другим средствам связи. Первоначально теория была посвящена каналу связи, определяемому длиной волны электромагнитного излучения или колебаний воздуха

Базовые понятия теории информации.

Информация – нематериальная сущность, при помощи которой с любой точностью можно описывать реальные (материальные), виртуальные(возможные) и понятийные сущности. Иначе, **информация** - сведения об объектах и явлениях окружающей среды, их параметрах, свойствах и состоянии, которые уменьшают имеющуюся о них степень неопределенности, неполноты знаний.

Теория информации (математическая теория связи) — раздел прикладной математики, аксиоматически определяющий понятие информации[1], её свойства и устанавливающий предельные соотношения для систем передачи данных.

Использует, главным образом, математический аппарат теории вероятностей и математической статистики.

Основные разделы теории информации — кодирование источника (сжимающее кодирование) и канальное (помехоустойчивое) кодирование. Теория информации тесно связана с криптографией и другими смежными дисциплинами.

Канал связи – это среда передачи информации, которая характеризуется в первую очередь максимально возможной для нее скоростью передачи данных(емкостью канала связи).

Шум – это помехи в канале связи при передаче информации.

Кодирование – преобразование дискретной информации одним из следующих способов: шифрование, сжатие, защита от шума.

Информатика рассматривает информацию как концептуально связанные между собой сведения, данные, понятия, изменяющие наши представления о явлении или объекте окружающего мира. Наряду с информацией в информатике часто употребляется понятие *данные*. Покажем, в чем их отличие.

Данные могут рассматриваться как признаки или записанные наблюдения, которые по каким-то причинам не используются, а только хранятся. В том случае, если появляется возможность использовать эти данные для уменьшения неопределенности о чем-либо, данные превращаются в информацию.

Формы адекватности информации

Адекватность информации - это определенный уровень соответствия создаваемого с помощью полученной информации образа реальному объекту, процессу, явлению и т.п.

Адекватность информации может выражаться в трех формах: семантической, синтаксической, прагматической.

Синтаксическая адекватность. Она отображает формально-структурные характеристики информации и не затрагивает ее смыслового содержания. На синтаксическом уровне учитываются тип носителя и способ представления информации, скорость передачи и обработки, размеры кодов представления информации, надежность и точность преобразования этих кодов и т.п. Информацию, рассматриваемую только с синтаксических позиций, обычно называют данными, так как при этом не имеет значения смысловая сторона. Эта форма способствует восприятию структурных внешних характеристик, т.е. синтаксической стороны информации.

Семантическая (смысловая) адекватность. Эта форма определяет степень соответствия образа объекта и самого объекта. Семантический аспект предполагает учет смыслового содержания информации. На этом уровне анализируются те сведения, которые отражает информация, рассматриваются смысловые связи. В информатике устанавливаются смысловые связи между кодами представления информации.

Эта форма служит для формирования понятий и представлений, выявления смысла, содержания информации и ее обобщения.

Прагматическая (потребительская) адекватность. Она отражает отношение информации и ее потребителя, соответствие информации цели управления, которая на ее основе реализуется. Проявляются прагматические свойства информации только при наличии единства информации (объекта), пользователя и цели управления. Прагматический аспект рассмотрения связан с ценностью, полезностью использования информации при выработке потребителем решения для достижения своей цели.

Меры информации

Классификация мер

Для измерения информации вводятся два параметра: количество информации *I* и объем данных *Vд.*

Эти параметры имеют разные выражения и интерпретацию в зависимости от рассматриваемой формы адекватности. Каждой форме адекватности соответствует своя мера количества информации и объема данных.

Синтаксическая мера информации

Эта мера количества информации оперирует с обезличенной информацией, не выражающей смыслового отношения к объекту.

Объем данных Vд в сообщении измеряется количеством символов (разрядов) в этом сообщении. В различных системах счисления один разряд имеет различный вес и соответственно меняется единица измерения данных: в двоичной системе счисления единица измерения - бит *(bit — binary digit*—двоичный разряд);

в десятичной системе счисления единица измерения - дит (десятичный разряд).

Количество информации I на синтаксическом уровне невозможно определить без рассмотрения понятия неопределенности состояния системы (энтропии системы).

1865 г. немецкий физик Рудольф Клазиус ввел в статистическую физику понятие энтропии как меры уравновешенности системы. В 1921 году основатель большей части математической статистики, англичанин Роналд Фишер впервые ввел термин «информация» в математику, но полученные им формулы носят очень специальный характер.

В 1948 году Клод Шеннон в своих работах по теории связи выписывает формулы для вычисления количества информации и энтропии. Термин «энтропия» используется Шенноном по совету фон Неймана, отметившего совпадение полученных Шенноном формул с соответствующими формулами статистической физики.

Информация через неопределенность

Действительно, получение информации о какой-либо системе всегда связано с изменением степени неосведомленности получателя о состоянии этой системы. Рассмотрим это понятие.

Пусть до получения информации потребитель имеет некоторые предварительные (априорные) сведения о системе α. Мерой его неосведомленности о системе является функция *H(α),* которая в то же время служит и мерой неопределенности состояния системы.

После получения некоторого сообщения β получатель приобрел некоторую дополнительную информацию $I_\beta(\alpha)$, уменьшившую его априорную неосведомленность гак, что апостериорная (после получения сообщения β) неопределенность состояния системы стала $H_\beta(\alpha)$.

Тогда количество информации $I_\beta(\alpha)$ о системе, полученной в сообщении β, определится как $I_\beta(\alpha) = H(\alpha) - H_\beta(\alpha)$, т.е. количество информации измеряется изменением (уменьшением) неопределенности состояния системы.

Если конечная неопределенность $H_\beta(\alpha)$ обратится в нуль, то первоначальное неполное знание заменится полным знанием и количество информации $I_\beta(\alpha) = H(\alpha)$.

Иными словами *энтропия системы* H(α) может рассматриваться как мера недостающей информации.

Энтропия системы H(α), имеющая *N* возможных состояний, согласно формуле Шеннона, равна:

$$H(\alpha) = -\sum_{i=1}^{N} P_i \log P_i$$

где P_i - вероятность того, что система находится в i-м состоянии.

Для случая, когда все состояния системы равновероятны, т.е. их вероятности равны,

$P_i = \frac{1}{N}$ ее энтропия определяется соотношением.

$$H(\alpha) = -\sum_{i=1}^{N} \frac{1}{N} \log \frac{1}{N}$$

Часто информация кодируется числовыми кодами в той или иной системе счисления, особенно это актуально при представлении информации в компьютере.

Естественно, что одно и то же количество разрядов в разных системах счисления может передать разное число состояний отображаемого объекта, что можно представить в виде соотношения

$$N=m^n,$$

где *N* — число всевозможных отображаемых состояний;

m - основание системы счисления (разнообразие символов, применяемых в алфавите);

n - число разрядов (символов) в сообщении.

Наиболее часто используются двоичные и десятичные логарифмы. Единицами измерения в этих случаях будут соответственно бит и дит.

Теорема Хартли: Информативность символа m-элементного алфавита равна *log m.*

Формула Шеннона выражает информативность источника информации с *m*-символьным алфавитом и данной частотной характеристикой.

Но каналу связи передается n-разрядное сообщение, использующее *m* различных символов. Так как количество всевозможных кодовых комбинаций будет $N=m^n$, то при равновероятности появления любой из них количество информации, приобретенной абонентом в результате получения сообщения, будет $I=logN=logm$ - формула Хартли.

Если в качестве основания логарифма принять *m,* то $I=n$. В данном случае количество информации (при условии полного априорного незнания абонентом содержания сообщения) будет равно объему данных $I=V_д$, полученных по каналу связи. Для неравновероятных состояний системы всегда $I<V_д=n$.

Коэффициент (степень) информативности (лаконичность) сообщения определяется отношением количества информации к объему данных, т.е.

$$Y = \frac{I}{V_д}, \text{ причем } Y<0<1,$$

С увеличением *Y* уменьшаются объемы работы по преобразованию информации (данных) в системе. Поэтому стремятся к повышению информативности, для чего разрабатываются специальные методы оптимального кодирования информации.

Формулы Шеннона

Равномерное распределение имеет наибольшую энтропию среди всех распределений с данным числом исходов.

Предложенный Шенноном способ измерения количества информации, содержащейся в одной случайной величине относительно другой случайной величины лежит в основе теории информации. Он приводит к числовой записи количества информации.

Для ДСВ(дискретных случайных величин) X и Y, заданных законами распределения $P(X = X_i) = p_i$, $P(Y = Y_i) = q_j$ и совместным распределением $P(X = X_i, Y = Y_i) = p_{ij}$, количество информации, содержащейся в X относительно Y, равно

$$I(X,Y) = \sum_{i,j} p_{ij} \log_2 \frac{p_{ij}}{p_i q_j}$$

Для НСВ(непрерывных случайных величин) X и Y, заданных плотностями распределения вероятностей px(t1), py(t2) и pxy(t1, t2), аналогичная формула имеет вид

$$I(X,Y) = \iint\limits_{\mathbb{R}^2} p_{XY}(t_1,t_2) \log_2 \frac{p_{XY}(t_1,t_2)}{p_X(t_1)p_Y(t_2)} dt_1 dt_2.$$

Очевидно, что

$$P(X = X_i, X = X_j) = \begin{cases} 0, & \text{при } i \neq j \\ P(X = X_i), & \text{при } i = j \end{cases}$$

и, следовательно,

$$I(X,X) = \sum_i p_i \log_2 \frac{p_i}{p_i p_i} = -\sum_i p_i \log_2 p_i.$$

Энтропия д.с.в. X в теории информации определяется формулой

$$H(X) = HX = I(X,X).$$

Свойства меры информации и энтропии:

1) $I(X,Y) \geqslant 0$, $I(X,Y) = 0 \quad \Leftrightarrow \quad X$ и Y независимы;
2) $I(X,Y) = I(Y,X)$;
3) $HX = 0 \quad \Leftrightarrow \quad X$ — константа;
4) $I(X,Y) = HX + HY - H(X,Y)$, где $H(X,Y) = -\sum_{i,j} p_{ij} \log_2 p_{ij}$;
5) $I(X,Y) \leqslant I(X,X)$. Если $I(X,Y) = I(X,X)$, то X — функция от Y.

Пример использования энтропии Шеннона.

Энтропия ДСВ – это минимум среднего количества бит, которое нужно передавать по каналу связи о текущем значении данной ДСВ.

Рассмотрим пример (скачки). В заезде участвуют 4 лошади с равными шансами на победу. Введем ДСВ X, равную номеру победившей лошади. Здесь HX = 2. После каждого заезда по каналам связи достаточно будет передавать два бита информации о номере победившей лошади. Кодируем номер лошади следующим образом: 1-00, 2-01, 3-10, 4-11. Если ввести функцию L(X), которая возвращает длину сообщения, кодирующего заданное значение X, то м.о. ML(X) – это средняя длина сообщения, кодирующего X. Можно формально определить L через две функции L(X) = len(code), где code(X) каждому значению X ставит в соответствие некоторый битовый код, причем, взаимно однозначно, а len возвращает длину в битах для любого конкретного кода. В этом примере ML(X)=HX.

Пусть теперь ДСВ X имеет следующее распределение

$$P(X=1) = \frac{3}{4}, P(X=2) = \frac{1}{8}, P(X=3) = P(X=4) = \frac{1}{16}$$

Тогда $HX = \frac{3}{4}\log_2 \frac{4}{3} + \frac{1}{8}\log_2 \frac{8}{1} + \frac{1}{8}\log_2 \frac{16}{1} = 1.186 \frac{\text{бит}}{\text{сим}}$

Закодируем номера лошадей: 1-0, 2-10, 3-110, 4-111, - т.е. так, чтобы каждый код не был префиксом другого кода (такое кодирование называется префиксным). В среднем в 16 заездах 1-я лошадь должна победить 12 из них, 2-я – в 2-х, 3-я в 1-м и 4-я – в одном. Таким образом, средняя длина сообщения о победителе равна (1*12+2*2+3*1+3*1)/16 = 1.375 бит/сим. Или м.о. L(X). Действительно, L(X) сейчас задается следующим распределением вероятностей: P(L(X)=1) = ¾, P(L(X)=2) = 1/8, P(L(X)=3) = 1/8. Следовательно,

M(L(X)) = ¾+2/8+3/8=11/8 = 1.375 бит/сим.

Таким образом, ML(X)>HX.

Может быть доказано, что более эффективного кодирования для рассмотренного случая не существует.

То, что энтропия Шеннона соответствует интуитивному представлению о мере информации, может быть продемонстрировано в опыте по определению среднего времени психических реакций. Опыт заключается в том, что перед испытуемым человеком зажигается одна из N-лампочек, которую он должен указать, Проводится большая серия испытаний, в которых каждая лампочка зажигается с определенной вероятностью. Оказывается, среднее время, необходимое для правильного ответа испытуемого, пропорционально величине энтропии ($-\sum_{i=1}^{N} p_i \log_2 p_i$), а не числу лампочек, как можно было подумать. В этом опыте предполагается, что чем больше информации будет получено человеком, тем дольше будет время ее обработки и, соответственно, реакции на нее.

Семантическая мера информации

Для измерения смыслового содержания информации, т.е. ее количества на семантическом уровне, наибольшее признание получила тезаурусная мера, которая связывает семантические свойства информации со способностью пользователя принимать поступившее сообщение. Для этого используется понятие *тезаурус пользователя.*

Тезаурус - это совокупность сведений, которыми располагает пользователь или система.

В зависимости от соотношений между смысловым содержанием информации *S* и тезаурусом пользователя S_P изменяется количество семантической информации 1_C, воспринимаемой пользователем и включаемой им в дальнейшем в свой тезаурус.

Характер такой зависимости показан на рис. 2. Рассмотрим два предельных случая, когда количество семантической информации I_C равно 0:

- при $S_P = 0$ пользователь не воспринимает, не понимает поступающую информацию;
- при $S_P \rightarrow \infty$ пользователь все знает, и поступающая информация ему не нужна.

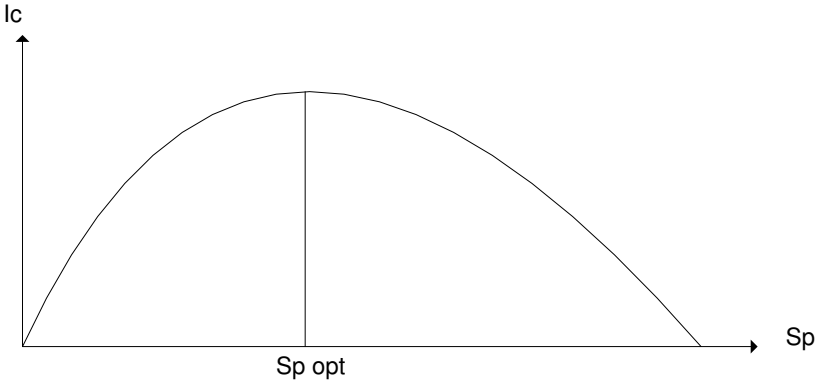


Рис. 2. Зависимость количества семантической информации, воспринимаемой потребителем, от его тезауруса $I_C = f(S_H)$

Максимальное количество семантической информации I_c потребитель приобретает при согласовании ее смыслового содержания *S* со своим тезаурусом S_P ($S_P = S_{P\,opt}$), когда поступающая информация понятна пользователю и несет ему ранее не известные (отсутствующие в его тезаурусе) сведения.

Следовательно, количество семантической информации в сообщении, количество новых знаний, получаемых пользователем, является величиной относительной. Одно и то же сообщение может иметь смысловое содержание для компетентного пользователя и быть бессмысленным (семантический шум) для пользователя некомпетентного.

При оценке содержательного аспекта информации необходимо стремиться к согласованию величин *S* и S_P.

Относительной мерой количества семантической информации может служить коэффициент содержательности С, который определяется как отношение количества семантической информации к ее объему:

$$C = \frac{I_C}{V_Д}$$

Функции меры семантической информации.

В 50-х годах XX века появились первые попытки определения абсолютного информационного содержания предложений естественного языка. Шеннон однажды заметил, что смысл сообщений не имеет никакого отношения к его теории информации, целиком построенной на положениях теории вероятностей. Но его способ точного измерения информации наводил на мысль о возможности существования способов точного измерения информации более общего вида, например, информации из

предложений естественного языка. Примером одной из таких мер является функция *inf(s)* = -$\log_2$p(s), где *s* - это предложение, смысловое содержание которого измеряется, *p(s)* - вероятность истинности s. Вот некоторые свойства этой функции-меры:

1) если s_1 => s_2 (из s_1 следует s_2) — истинно, то *inf*(s_1) >= inf(s_2);
2) *inf(s)>=0;*
3) если s — истинно, то *inf(s)* = 0;
4) *inf*(s_1s_2) = *inf*(s_1) + inf(s_2) <=> *p*(s1 s2) = *p(s1)p(s_2),* т.е. независимости s_1 и s_2.

Значение этой функции-меры больше для предложений, исключающих большее количество возможностей. Пример: из s_1 — "a > 3" и s_2 — "a = *7'* следует, что s_2 => s_1 или *inf(s_2)* >= inf(s_1); ясно, что s_2 исключает больше возможностей, чем s_1.

Прагматическая мера информации

Эта мера определяет полезность информации (ценность) для достижения пользователем поставленной цели. Эта мера также величина относительная, обусловленная особенностями использования этой информации в той или иной системе.

Качество информации

Возможность и эффективность использования информации обусловливаются такими основными ее потребительскими *показателями качества,* как репрезентативность, содержательность, достаточность, доступность, актуальность, своевременность, точность, достоверность, устойчивость.

Репрезентативность информации связана с правильностью ее отбора и формирования в целях адекватного отражения свойств объекта. Важнейшее значение здесь имеют:

- правильность концепции, на базе которой сформулировано исходное понятие;
- обоснованность отбора существенных признаков и связей отображаемого явления.

Нарушение репрезентативности информации приводит нередко к существенным ее погрешностям.

Содержательность информации отражает семантическую емкость, равную отношению количества семантической информации в сообщении к объему обрабатываемых данных, т.е.

$$C = \frac{I_C}{V_Д}$$

С увеличением содержательности информации растет семантическая пропускная способность информационной системы, так как для получения одних и тех же сведений требуемся преобразовать меньший объем данных.

Наряду с коэффициентом содержательности С, отражающим семантический аспект, можно использовать и коэффициент информативности, характеризующийся отношением количества синтаксической информации (по Шеннону) к объему данных

$$Y = \frac{I}{V_Д}$$

Достаточность (полнота) информации означает, что она содержит минимальный, но достаточный для принятия правильного решения состав (набор показателей).

Понятие полноты информации связано с ее смысловым содержанием (семантикой) и прагматикой. Как неполная, т.е. недостаточная для принятия правильного решения, так и избыточная информация снижает эффективность принимаемых пользователем решений.

Доступность информации восприятию пользователя обеспечивается выполнением соответствующих процедур ее получения и преобразования. Например, в информационной системе информация преобразовывается к доступной и удобной для восприятия пользователя форме. Это достигается, в частности, и путем согласования ее семантической формы с тезаурусом пользователя.

Актуальность информации определяется степенью сохранения ценности информации для управления в момент ее использования и зависит от динамики изменения ее характеристик и от интервала времени, прошедшего с момента возникновения данной информации**.**

Своевременность информации означает ее поступление не позже заранее назначенного момента времени, согласованного со временем решения поставленной задачи.

Точность информации определяется степенью близости получаемой информации к реальному состоянию объекта, процесса, явления и т.п.

Достоверность информации определяется ее свойством отражать реально существующие объекты с необходимой точностью. Измеряется достоверность информации доверительной вероятностью необходимой точности, т.е. вероятностью того, что отображаемое информацией значение параметра отличается от истинного значения этого параметра в пределах необходимой точности.

Устойчивость информации отражает ее способность реагировать на изменения исходных данных без нарушения необходимой точности.

Источники

1. Лидовский, Теория информации, 2004. Разделы 5,7,8,9.

2. Конспект лекций УГТУ по информатике стр. 10-17

Лекция 2. ИТ, системный подход, информационные системы.

Информационные потоки, модели и их описание. Понятие информационной системы. Системный подход к построению информационных систем.

Информационные технологии. Программная, аппаратная и человеческие части ИС. Разновидности ИС.

Оглавление

Информационные технологии.

Информационная модель — модель объекта, представленная в виде информации, описывающей существенные для данного рассмотрения параметры объекта, связи между ними, входы и выходы объекта и позволяющая путём подачи на модель информации об изменениях входных величин моделировать возможные состояния объекта.

Информационная модель — формальная модель ограниченного набора фактов, понятий или инструкций, предназначенная для удовлетворения конкретному требованию (ИСО 10303-1:1994, статья 3.2.21).

Информационная технология - совокупность методов, производственных и программно-технологических средств, объединенных в технологическую цепочку, обеспечивающую сбор, хранение, обработку, вывод и распространение информации.

Инфраструктурные свойства информационных технологий.

[Николас Карр. Does IT matter, 50 – 51, 17-18]

Николас Карр, бывший редактор Harvard Business Review, в издании «Does IT matter» отмечает инфраструктурную специфику информационных технологий, сравнивая их с железнодорожным транспортом или линиями телеграфа.

Инфраструктурная технология, проходит определенные этапы своего развития:

1) Становление. Высокая стоимость, долговременное преимущество владельца технологии.
2) Активный рост. Развитие технологии в количественном (распространение) и качественном (стандартизация) виде.
3) Общедоступность. Низкая стоимость, технология является необходимой но не достаточной составляющей конкурентного преимущества.

В качестве примера развития информационных технологий рассмотрим приводимый Карром пример систем бронирования авиабилетов.

Сначала доступ к ИТ был ограничен техническими препятствиями. Поскольку до 1950-х годов бизнес-компьютеров не существовало вообще, компания, которая хотела обладать таковыми, должна была создавать их сама.

Разумеется, трудность создания информационных систем уже сама по себе означала, что любая компания, совершившая такой «подвиг», значительно опережала конкурентов. Чтобы повторить прорыв в обработке корпоративной информации, последним могли потребоваться долгие годы. Пожалуй, самым известным примером компании, получившей преимущество первопроходца, служит авиакомпания American Airlines с ее системой бронирования авиабилетов Sabre. В 1953 году авиаперевозчик начал переговоры с IBM о возможности создания автоматизированной системы бронирования билетов. В то время эта громоздкая процедура выполнялась в основном вручную, что требовало больших затрат труда и вело к множеству ошибок. Информация о наличии посадочных мест хранилась отдельно от данных о пассажирах, что требовало сложной перепроверки, которая повышала издержки и была чревата новыми ошибками. Для обработки всех этих данных каждой крупной авиакомпании требовался особый большой офис, обстановка в котором напоминала военные действия. Хотя для распределения посадочных мест компания использовала простейшую механическую систему под названием Reservisor, она по-прежнему в значительной степени зависела от традиционных трудоемких операций, выполняемых вручную.

Авиакомпания понимала, что совершенствование системы бронирования билетов может дать колоссальные конкурентные преимущества. Прежде всего автоматизированная система значительно снизила бы затраты на рабочую силу. Во-вторых, уменьшение ошибок позволило бы сократить резервный запас свободных мест на

каждом рейсе и тем самым заметно повысить прибыль. В-третьих, превращение бронирования билетов в надежную и легкую процедуру сделало бы рейсы компании более привлекательными для потребителей. Наконец, централизованная автоматизированная система позволила бы компании тщательнее анализировать свои операции, что повысило бы эффективность принятия решений относительно разработки маршрутов, использования авиапарка, предоставляемых услуг и тарифов.

Хотя в середине 1950-х уровень развития техники и ПО не позволял создавать такие сложные системы, работающие в режиме реального времени, становилось все более очевидным, что необходимая технология скоро появится. В 1959 году после шести лет предварительных исследований президент компании American Airlines С. Р. Смит (C. R. Smith) подписал контракт на разработку необходимого программного обеспечения, которое должно было работать сразу на двух мейнфреймах IBM 7090. Это было масштабное и рискованное начинание, потребовавшее пяти лет работы двухсот квалифицированных инженеров и техников. Оно обошлось American Airlines примерно в $30 млн, что по тем временам было огромной суммой.

Однако когда в 1962 году начались первые пробные запуски системы, сразу же стало ясно, что Sabre может реализовать свой потенциал и создать авиакомпании конкурентное преимущество. Рост эффективности был колоссальным: на обслуживание того количества операций, на которое служащим компании требовался целый день, Sabre тратила всего несколько минут. В то же время частота появления ошибок упала с 8% до менее чем 1% [15]. Как и ожидалось, компьютерная обработка информации позволила American Airlines повысить гибкость размещения ресурсов и эффективность ценообразования. Анализ финансовых результатов показал, что окупаемость масштабных инвестиций составила 25% [16]. Столь же огромным был выигрыш в области маркетинга. Корреспондент The Wall Street Journal Томас Петцингер (Thomas Petzinger) в книге «Жесткая посадка» (Hard Landing) пишет: «Почти мгновенно компания American Airlines начала увеличивать свою долю на рынке за счет других авиакомпаний, включая своего главного конкурента — United Airlines. После этого любая авиакомпания, игнорировавшая компьютерную революцию, подвергала себя большой опасности» [17].

Разумеется, лишь немногие авиакомпании действительно игнорировали компьютерную революцию. Большинство конкурентов American Airlines быстро оценили полученное ей преимущество и немедленно начали разрабатывать собственные системы бронирования билетов. Компания IBM в свою очередь была рада помогать им. Опираясь на использование опыта, накопленного при разработке Sabre, компьютерный гигант создал типовую систему PARS, которую весьма успешно продавал другим авиакомпаниям. К началу 1970-х годов ряд систем, основанных на PARS, и в первую очередь Apollo авиакомпании United Airlines, считались технически более совершенными, чем Sabre. Однако преодолеть отставание от American Airlines было непросто. К концу 1970-х компании удалось сделать Sabre основной системой бронирования билетов, используемой туристическими агентствами. Появился новый важный источник прибыли и маркетинговое преимущество на высококонкурентных маршрутах.

Если компания не в состоянии удержать технологическое преимущество в течение сколько-нибудь значительного времени, ее стратегия первопроходца может привести к плачевному результату. Конкуренты не просто догонят ее. Они ее перегонят, создав более мощную систему.

ИТ не только трансформировали отдельные бизнес-процессы, например обработку заказов, но и изменили целые отрасли и создали новые. Как показывает история компании Reuters, дальновидность и здесь давала огромные конкурентные преимущества. Со времени основания в середине XIX века Reuters была пионером в области технологий связи. Когда компания впервые вышла на рынок в 1849 году, она использовала «низкую технологию», а именно почтовых голубей, которые доставляли биржевые котировки в районы между Брюсселем и Аахеном, где не было телеграфа. (Дело в том, что бельгийская

телеграфная линия заканчивалась в Брюсселе, а немецкая начиналась только в Аахене.) Два года спустя компания Reuters стала телеграфным агентством, продававшим информацию о ценах. Она использовала новый кабель, проложенный через Ла-Манш и соединявший Лондон с Парижем. В начале XX века агентство одним из первых начало использовать для передачи новостей радио и телетайп, а в 1964 году — применять компьютеры для ускорения передачи финансовой информации.

Но самый большой успех в области технологии, по всей видимости, был достигнут в начале 1970-х. Это были последние годы существования Бреттон-Вудской валютной системы с фиксированным курсом валют, введенной в 1944 году. Компания Reuters поняла, что свободное колебание курсов валют приведет к появлению чувствительного валютного рынка. Это потребует чрезвычайно быстрой передачи информации о курсах валют и валютных торгах. Телефоны и телексы, которыми традиционно пользовались дилеры, не смогут передавать большие объемы информации с достаточной скоростью.

Компания попыталась оторваться от конкурентов, разработав принципиально новую услугу — Reuter Monitor Money Rates. Для этого в банках, офисах компаний и у других дилеров были установлены специальные терминалы. Фактически компания создала и контролировала систему электронных торгов. Собственная сеть стала основным инструментом валютных торгов, благодаря которому компания Reuters получила новый важный источник доходов и прибыли. Сеть компании также послужила стартовой площадкой для запуска многих новых услуг в области передачи информации (биржевые котировки, сводки новостей и т. д.). Тем самым была заложена основа для быстрого роста прибыли в течение двух десятилетий. В 1980-е годы прибыль Reuters (без вычета налогов) возросла с 3,9 млн до 283,1 млн фунтов стерлингов [21].

Время, необходимое конкурентам на копирование новой технологии, или так называемый **репликативный цикл технологии**, является важнейшим показателем, используемым при оценке стратегических перспектив инвестиций в ИТ. История ИТ подтверждает одну общую истину: с течением времени репликативный цикл технологии становится все короче и короче. По мере совершенствования компьютерной техники и программного обеспечения, снижения их стоимости и распространения знаний о них конкуренты все быстрее осваивают потенциальные и реальные возможности новых систем.

Гомогенизация процессов.

Существует еще одна аналогия между развитием ИТ и предшествующих инфраструктурных технологий. Развитие ИТ-инфраструктуры привело к стандартизации не только самой технологии, но в значительной мере и способов ее применения. Производители ПО сегодня внедряют последние достижения в области деловой практики в разрабатываемые ими программы. Действительно, можно утверждать, что по мере совершенствования бизнес-систем конкуренция между поставщиками разворачивается не столько вокруг самой технологии, сколько вокруг использования передового опыта ее применения в отрасли.

Это касается прежде всего автоматизации систем предприятия. В отличие от первых пакетов ПО, позволяющих автоматизировать определенные виды деятельности (например, выписывание и оформление счетов), корпоративные системы автоматизируют процесс в целом (чаще всего важнейшие бизнес-процессы предприятия). В то же время программное обеспечение налагает на процесс автоматизации определенные ограничения. Оно оказывает важное и даже определяющее влияние на то, как протекает этот процесс. Например, покупая у Microsoft пакет приложений для управления отношениями с клиентами, компания одновременно покупает и философию подхода Microsoft к работе с клиентами. ПО и подразумеваемая бизнес-практика становятся неразличимыми.

В статье о системах планирования ресурсов предприятия (ERP), опубликованной в 1998 году в Harvard Business Review, исследователь ИТ Томас Девенпорт (Thomas

Davenport) так объясняет это явление: «При разработке в прошлом информационных систем компании сначала решали, как они намерены вести бизнес, а затем выбирали устраивающий их пакет ПО, который должен был поддерживать фирменные бизнес-процессы. Чтобы система полностью соответствовала бизнес-процессам, они часто переписывали большие куски программы. Однако с появлением интегрированных систем управления предприятиями эта последовательность изменилась. Часто приходится менять процесс, подгоняя его под систему».

Инфраструктурный характер ИТ говорит не об их низкой значимости, а о высокой динамике распространения и смещении акцентов к методам внедрения технологий.

[Инь и Янь, введение].

В России с начала 90-х годов шел процесс насыщения организаций техническими средствами, созданию локальных и глобальных сетей. Это является предпосылкой осмысления архитектуры организаций с позиции применения комплексных информационных систем. Без наличия архитектуры предприятия невозможно обеспечить развитие ИТ в организациях, управлять инвестициями в ИТ. Результатом отсутствия такой архитектуры может стать то, что организация будет иметь разобщенные операции и системы, что, в свою очередь, приведет к бессмысленному дублированию, несовместимости и дополнительным затратам.

Системный подход, системный анализ.

Система.

[Сурьмин – Теория систем и системный анализ, 55-57].

Несмотря на огромный теоретический задел, наблюдается неоднозначность понимания категории “система”. Основатель теории систем австриец Людвиг фон Берталанфи рассматривал систему как комплекс взаимодействующих элементов. Это понятие до сих пор — основа используемых понятий “системы”. Берталанфи сделал особый акцент не на том, что целое состоит из частей, а на том, что поведение и свойства целого определяются взаимодействием его частей. Однако подходом к объекту как к комплексу взаимодействующих частей понимание системы не исчерпывается. Существуют и другие характеристики.

По формам движения можно выделить механические, физические, химические, биологические, социальные и прочие системы. Так как высшая форма движения включает в себя низшие, то системы помимо их специфических свойств имеют общие свойства, не зависящие от их природы. Эта общность свойств и позволяет определять понятием “система” самые разнородные совокупности.

Понятие “система” обладает двумя противоположными свойствами: ***ограниченностью и целостностью***. Первое — это внешнее свойство системы, а второе — внутреннее, приобретаемое в процессе развития. Система может быть отграниченной, но не целостной (например, недостроенный дом), но, как правило, чем более система выделена, отграничена от среды, тем более она внутренне целостна.

Согласно вышесказанному можно дать определение системы как отграниченного, взаимно связанного множества, отражающего объективное существование конкретных отдельных взаимосвязанных совокупностей объектов и не содержащего специфических ограничений, присущих частным системам.

Важнейшие свойства системы: ***структурность***, ***взаимозависимость со средой, иерархичность, множественность описаний*** (Таблица 1).

Таблица 1 Свойства системы

Свойство системы	Характеристика

Ограниченность	Система отделена от окружающей среды границами
Целостность	Ее свойство целого принципиально не сводится к сумме свойств составляющих элементов
Структурность	Поведение системы обусловлено не только особенностями отдельных элементов, сколько свойствами ее структуры
Взаимозависимость со средой	Система формирует и проявляет свойства в процессе взаимодействия со средой
Иерархичность	Соподчиненность элементов в системе
Множественность описаний	По причине сложности познание системы требует множественности ее описаний

Ограниченность системы представляет собой первое и изначальное ее свойство. Это необходимое, но не достаточное свойство. Если совокупность объектов ограничена от внешнего мира, то она может быть системной, а может и не быть ею.

Совокупность становится системой только тогда, когда она обретает **целостность**, т.е. приобретает **структурность**, **иерархичность**, **взаимосвязь со средой**. Целостность не сводима к своим составным частям. При этом всегда наблюдается потеря качества.

Поскольку научное описание объекта предполагает процедуры мысленного расчленения целостности, то целостность представляет собой некоторое множество описаний.

Отсюда многообразие определений системы: структурированное множество; множество, взаимодействующее с окружением; упорядоченная целостность и т.д.

Системный подход.

[Сурьмин – Теория систем и системный анализ, 3-9]

Системный подход состоит в том, что любой более или менее сложный объект рассматривается в качестве относительно самостоятельной системы со своими особенностями функционирования и развития.

Системное познание и преобразование мира предполагают:

1. рассмотрение объекта деятельности (теоретической и практической) как системы, т.е. как ограниченного множества взаимодействующих элементов;
2. установление состава, структуры и организации элементов и частей системы, обнаружение ведущих взаимодействий между ними;
3. выявление внешних связей системы, выделение главных;
4. определение функций системы и ее роли среди других систем;
5. анализ противоречий структуры и функций системы;
6. обнаружение на этой основе закономерностей и тенденций развития системы.

Информационная система

Информационная система - взаимосвязанная совокупность средств, методов и персонала, используемых для хранения, обработки и выдачи информации в интересах достижения поставленной цели.
[Вендеров CASE-технологии, Введение]

Тенденции развития современных информационных технологий приводят к постоянному возрастанию сложности информационных систем (ИС). Современные крупные проекты ИС характеризуются, как правило, следующими особенностями:

- сложность описания (достаточно большое количество функций, процессов, элементов данных и сложные взаимосвязи между ними), требующая тщательного моделирования и анализа данных и процессов;
- наличие совокупности тесно взаимодействующих компонентов (подсистем), имеющих свои локальные задачи и цели функционирования (например, традиционных приложений, связанных с обработкой транзакций и решением регламентных задач, и приложений аналитической обработки (поддержки принятия решений), использующих нерегламентированные запросы к данным большого объема);
- ограниченная возможность использования каких-либо типовых проектных решений и прикладных систем;
- необходимость интеграции существующих и вновь разрабатываемых приложений;
- функционирование в неоднородной среде на нескольких аппаратных платформах;
- разобщенность и разнородность отдельных групп разработчиков по уровню квалификации и сложившимся традициям использования тех или иных инструментальных средств;
- существенная временная протяженность проекта, обусловленная, с одной стороны, ограниченными возможностями коллектива разработчиков, и, с другой стороны, масштабами организации-заказчика и различной степенью готовности отдельных ее подразделений к внедрению ИС.

Процессы в информационной системе

Процессы, обеспечивающие работу информационной системы любого назначения:

- ввод информации из внешних или внутренних источников;
- обработка входной информации и представление ее в удобном виде;
- вывод информации для представления потребителям или передачи в другую систему;
- обратная связь - это информация, переработанная людьми данной организации для коррекции входной информации.

Бизнес-процесс организации – более общее понятие, может включать в себя процессы информационной системы, но выходить за ее пределы.

Информационные потоки

Информационный поток — это совокупность циркулирующих в системе, между системой и внешней средой сообщений, необходимых для управления, анализа и контроля операций.

Передача и прием информационных потоков осуществляется с помощью носителей памяти человека, документа, магнитного носителя, устной речи и т.п. Носитель информации — это любое материальное средство, фиксирующее информацию. Информационный поток может объединять носители разных видов, например, состоять из бумажных и электронных носителей, которые дублируют или дополняют друг друга.

Структура информационных потоков определяет их однородность и неоднородность. **Однородные информационные потоки** характеризуются единым видом носителя, единой функциональной принадлежностью.

По периодичности информационные потоки делятся на **регулярные**, соответствующие регламентированной во времени передаче данных, и **оперативные** — обеспечивающие связь по требованию.

Виды информационных потоков

Информационный поток характеризуется следующими показателями:

- источник возникновения;
- направление движения потока;
- скорость передачи и приема;
- интенсивность потока и др.

Формирование информационных систем невозможно без исследования потоков в разрезе определенных показателей. Если скорость и интенсивность потока не важны, критичным может оказаться их тип (видео, звук, печатные документы определенных видов).

Состав и структура ИС

Структура информационных систем

Подсистема - это часть системы, выделенная по какому-либо признаку.

Общую структуру информационной системы можно рассматривать как совокупность подсистем независимо от сферы применения. В этом случае говорят о структурном признаке классификации, а подсистемы называют обеспечивающими. Таким образом, структура любой информационной системы может быть представлена совокупностью обеспечивающих подсистем.

Типы обеспечивающих подсистем:

- Информационное обеспечение
- Техническое обеспечение
- Математическое и программное обеспечение
- Организационное обеспечение
- Правовое обеспечение
- Программное обеспечение

Типы обеспечивающих подсистем

Информационное обеспечение

Информационное обеспечение - совокупность единой системы классификации и кодирования информации, унифицированных систем документации, схем информационных потоков, циркулирующих в организации, а также методология построения баз данных.

Назначение подсистемы информационного обеспечения состоит в своевременном формировании и выдаче достоверной информации для принятия управленческих решений.

Унифицированные системы документации создаются на государственном, республиканском, отраслевом и региональном уровнях. Главная цель - это обеспечение сопоставимости показателей различных сфер общественного производства.

Однако, несмотря на существование унифицированной системы документации, при обследовании большинства организаций постоянно выявляется целый комплекс типичных недостатков:

1. чрезвычайно большой объем документов для ручной обработки;
2. одни и те же показатели часто дублируются в разных документах;

3. работа с большим количеством документов отвлекает специалистов от решения непосредственных задач;
4. имеются показатели, которые создаются, но не используются, и др.

Поэтому устранение указанных недостатков является одной из задач, стоящих при создании информационною обеспечения.

Схемы информационных потоков отражают маршруты движения информации и ее объемы, места возникновения первичной информации и использования результирующей информации. За счет анализа структуры подобных схем можно выработать меры по совершенствованию всей системы управления.

Построение схем информационных потоков, позволяющих выявить объемы информации и провести ее детальный анализ, обеспечивает:

1. исключение дублирующей и неиспользуемой информации;
2. классификацию и рациональное представление информации.

При этом подробно должны рассматриваться вопросы взаимосвязи движения информации по уровням управления. Следует выявить, какие показатели необходимы для принятия управленческих решений, а какие нет. К каждому исполнителю должна поступать только та информация, которая используется.

Методология построения баз данных базируется на теоретических основах их проектирования. Для понимания концепции методологии приведем основные ее идеи в виде двух последовательно реализуемых на практике этапов:

1-й этап - обследование всех функциональных подразделений фирмы с целью:

1. понять специфику и структуру се деятельности;
2. построить схему информационных потоков;
3. проанализировать существующую систему документооборота;
4. определить информационные объекты и соответствующий состав реквизитов (параметров, характеристик), описывающих их свойства и назначение.

2-й этап - построение концептуальной информационно-логической модели данных для обследованной на 1-м этапе сферы деятельности. В этой модели должны быть установлены и оптимизированы все связи между объектами и их реквизитами. Информационно-логическая модель является фундаментом, на котором будет создана база данных.

Техническое обеспечение

Техническое обеспечение - комплекс технических средств, предназначенных для работы информационной системы, а также соответствующая документация на эти средства и технологические процессы.

Математическое и программное обеспечение

Математическое и программное обеспечение - совокупность математических методов, моделей, алгоритмов и программ для реализации целей и задач информационной системы.

В состав **программного обеспечения** входят общесистемные и специальные программные продукты, а также техническая документация.

К *общесистемному* программному обеспечению относятся комплексы программ, ориентированных на пользователей и предназначенных для решения типовых задач обработки информации. Они служат для расширения функциональных возможностей компьютеров, контроля и управления процессом обработки данных.

Специальное программное обеспечение представляет собой совокупность программ, разработанных при создании конкретной информационной системы. В его состав входят пакеты прикладных программ (ППП), реализующие разработанные модели разной степени адекватности, отражающие функционирование реального объекта.

Техническая документация на разработку программных средств должна содержать описание задач, задание на алгоритмизацию, экономико-математическую модель задачи, контрольные примеры.

Организационное обеспечение

Организационное обеспечение - совокупность методов и средств, регламентирующих взаимодействие работников с техническими средствами и между собой в процессе разработки и эксплуатации информационной системы.

Организационное обеспечение создается по результатам предпроектного обследования на этапе построения инфологических схем.

Правовое обеспечение

Правовое обеспечение - совокупность правовых норм, определяющих создание, юридический статус и функционирование информационных систем, регламентирующих порядок получения, преобразования и использования информации.

В состав правового обеспечения входят законы, указы, постановления государственных органов власти, приказы, инструкции и другие нормативные документы министерств, ведомств, организаций, местных органов власти. В правовом обеспечении можно выделить общую часть, регулирующую функционирование любой информационной системы, и специальную часть, регулирующую функционирование конкретной системы.

Правовое обеспечение этапов разработки информационной системы включает нормативные акты, связанные с договорными отношениями разработчика и заказчика и правовым регулированием отклонений от договора.

Правовое обеспечение этапов функционирования информационной системы включает:

- статус информационной системы;
- права, обязанности и ответственность персонала;
- правовые положения отдельных видов процесса управления;
- порядок создания и использования информации и др.

Разнообразие задач, решаемых с помощью ИС, привело к появлению множества разнотипных систем, отличающихся принципами построения и заложенными в них правилами обработки информации.

Классификация ИС

Информационные системы можно **классифицировать** по целому ряду различных признаков. В основу рассматриваемой классификации положены наиболее существенные признаки, определяющие функциональные возможности и особенности построения современных систем. В зависимости от объема решаемых задач, используемых технических средств, организации функционирования, информационные системы делятся на ряд групп (классов).

По типу хранимых данных ИС делятся на фактографические и документальные. Фактографические системы предназначены для хранения и обработки структурированных данных в виде чисел и текстов. Над такими данными можно выполнять различные операции. В документальных системах информация представлена в виде документов, состоящих из наименований, описаний, рефератов и текстов, изображений и образцов видеоряда. Поиск по неструктурированным данным осуществляется с использованием семантических признаков. Отобранные документы предоставляются пользователю, а обработка данных в таких системах практически не производится.

В настоящее время стремятся к переходу от документальных к фактографическим ИС.

Основываясь на **степени автоматизации информационных процессов** информационные системы делятся на ручные, автоматические и автоматизированные.

Рис. 1.1. Классификация информационных систем

Ручные ИС характеризуются отсутствием современных технических средств переработки информации и выполнением всех операций человеком.

В автоматических ИС все операции по переработке информации выполняются без участия человека.

Автоматизированные ИС предполагают участие в процессе обработки информации и человека, и технических средств, причем главная роль в выполнении рутинных операций обработки данных отводится компьютеру. Именно этот класс систем соответствует современному представлению понятия "информационная система".

В зависимости от **характера обработки** данных ИС делятся на **информационно-поисковые** и **информационно-решающие**.

Информационно-поисковые системы производят ввод, систематизацию, хранение, выдачу информации по запросу пользователя без сложных преобразований данных. (Например, ИС библиотечного обслуживания, резервирования и продажи билетов на транспорте, бронирования мест в гостиницах и пр.)

Информационно-решающие системы осуществляют, кроме того, операции переработки информации по определенному алгоритму. По **характеру использования выходной информации** такие системы принято делить на управляющие и советующие.

Результирующая информация управляющих ИС непосредственно трансформируется в принимаемые человеком решения. Для этих систем характерны задачи расчетного характера и обработка больших объемов данных. (Например, ИС планирования производства или заказов, бухгалтерского учета.)

Советующие ИС вырабатывают информацию, которая принимается человеком к сведению и учитывается при формировании управленческих решений. Эти системы имитируют интеллектуальные процессы обработки знаний, а не данных (например, экспертные системы).

Классификация в зависимости от **сферы применения** различают следующие классы ИС.

Информационные системы организационного управления - предназначены для автоматизации функций управленческого персонала.

Основными функциями подобных систем являются: оперативный контроль и регулирование, оперативный учет, перспективное и оперативное планирование, бухгалтерский учет, управление сбытом, снабжением и другие экономические и организационные задачи.

ИС управления технологическими процессами (ТП) - служат для автоматизации функций производственного персонала по контролю и управлению производственными операциями. В таких системах обычно предусматривается наличие развитых средств измерения параметров технологических процессов (температуры, давления, химического состава и т.п.), процедур контроля допустимости значений параметров и регулирования технологических процессов.

ИС автоматизированного проектирования (САПР) - предназначены для автоматизации функций инженеров-проектировщиков, конструкторов, архитекторов, дизайнеров при создании новой техники или технологии. Основными функциями подобных систем являются: инженерные расчеты, создание графической документации (чертежей, схем, планов), создание проектной документации, моделирование проектируемых объектов.

Интегрированные (корпоративные) ИС - используются для автоматизации всех функций организации и охватывают весь цикл работ от планирования деятельности до сбыта продукции. Они включают в себя ряд модулей (подсистем), работающих в едином информационном пространстве и выполняющих функции поддержки соответствующих направлений деятельности.

Анализ современного состояния рынка ИС показывает устойчивую тенденцию роста спроса на информационные системы организационного управления. Причем спрос

продолжает расти именно на интегрированные системы управления. Автоматизация отдельной функции, например, бухгалтерского учета или сбыта готовой продукции, считается уже пройденным этапом для многих предприятий.

Классификация ИС в зависимости от **уровня управления**, на котором система используется.

Информационная система оперативного уровня - поддерживает исполнителей, обрабатывая данные о событиях (счета, накладные, зарплата, кредиты, поток сырья и материалов). Информационная система оперативного уровня является связующим звеном между фирмой и внешней средой.

Задачи, цели, источники информации и алгоритмы обработки на оперативном уровне заранее определены и в высокой степени структурированы.

Информационные системы функционального уровня выполняют обработку данных и знаний, используются главным образом для контроля деятельности и поддержки работы специалистов.

Информационные системы специалистов - поддерживают работу с данными и знаниями, повышают продуктивность и производительность работы инженеров и проектировщиков. Задача подобных информационных систем - интеграция новых сведений в организацию и помощь в обработке бумажных документов.

Информационные системы уровня менеджмента - используются работниками среднего управленческого звена для мониторинга, контроля, принятия решений и администрирования. Основные функции этих информационных систем:

- сравнение текущих показателей с более ранними;
- составление периодических отчетов за определенное время, а не выдача отчетов по текущим событиям, как на оперативном уровне;
- обеспечение доступа к архивной информации и т.д.

Стратегическая информационная система - информационная система, обеспечивающая поддержку принятия решений по реализации стратегических перспективных целей развития организации.

Информационные системы стратегического уровня помогают высшему звену управленцев решать неструктурированные задачи, осуществлять долгосрочное планирование. Основная задача - сравнение происходящих во внешнем окружении изменений с существующим потенциалом фирмы. Они призваны создать общую среду компьютерной телекоммуникационной поддержки решений в неожиданно возникающих ситуациях. Используя самые совершенные программы, эти системы способны в любой момент предоставить информацию из многих источников. Некоторые стратегические системы обладают ограниченными аналитическими возможностями.

Источники

1. Грекул – Проектирование информационных систем 2005. Лекция 1.
2. Вендеров - CASE-технологии. Введение.
3. Сурьмин – Теория систем и системный анализ, 2003. (3-9, 55-57)
4. Николас Карр – Блеск и нищета информационных технологий. (17,18,50,51)
5. Данилин, Слюсаренко. Архитектура и стратегия. Инь и янь информационных технологий. Введение.

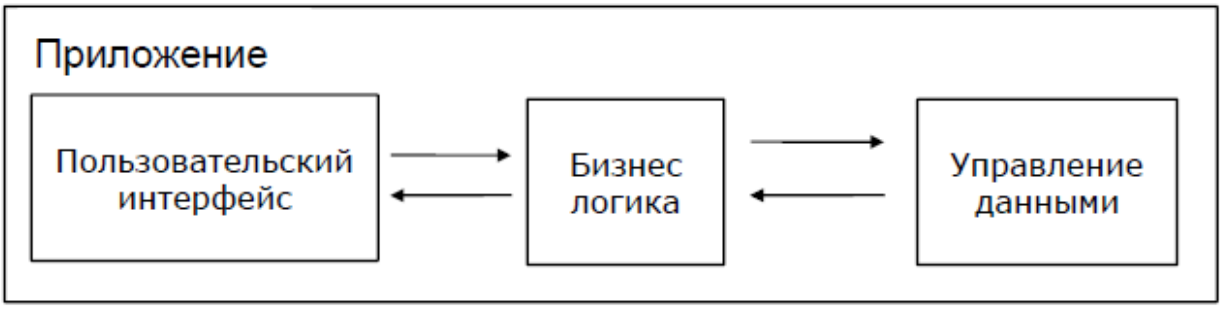
Лекция 3. Архитектура ИС.

Аннотация: Архитектура информационных систем.

Базовые функции информационных систем.

Архитектура информационной системы - концепция, определяющая модель, структуру, выполняемые функции и взаимосвязь компонентов информационной системы. (Глоссарий)

С точки зрения **программно-аппаратной реализации** можно выделить ряд типовых архитектур ИС.



Компоненты информационной системы по выполняемым функциям можно разделить на три слоя: слой представления, слой бизнес-логики и слой доступа к данным.

Слой представления - все, что связано с взаимодействием с пользователем: нажатие кнопок, движение мыши, отрисовка изображения, вывод результатов поиска и т.д.

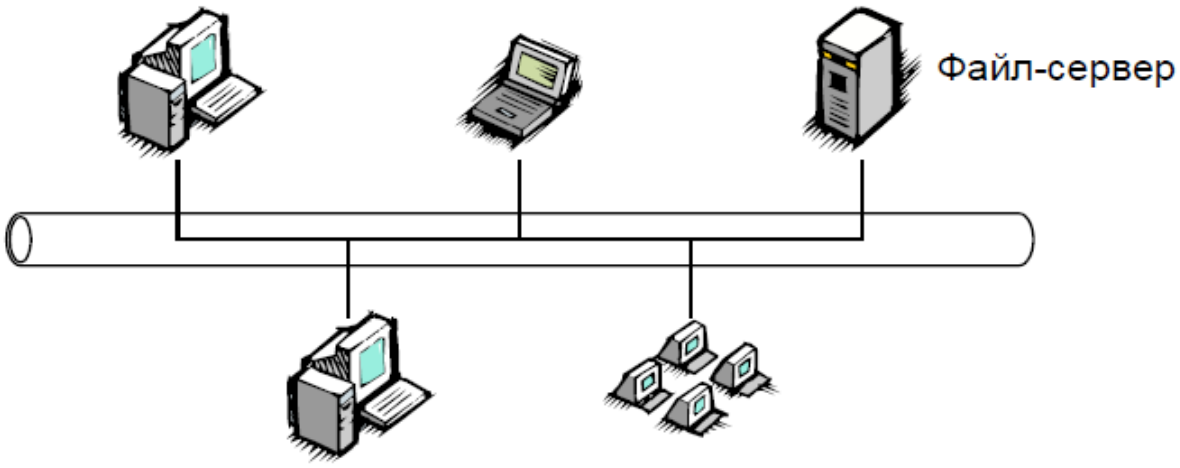
Бизнес логика - правила, алгоритмы реакции приложения на действия пользователя или на внутренние события, правила обработки данных.

Слой доступа к данным - хранение, выборка, модификация и удаление данных, связанных с решаемой приложением прикладной задачей

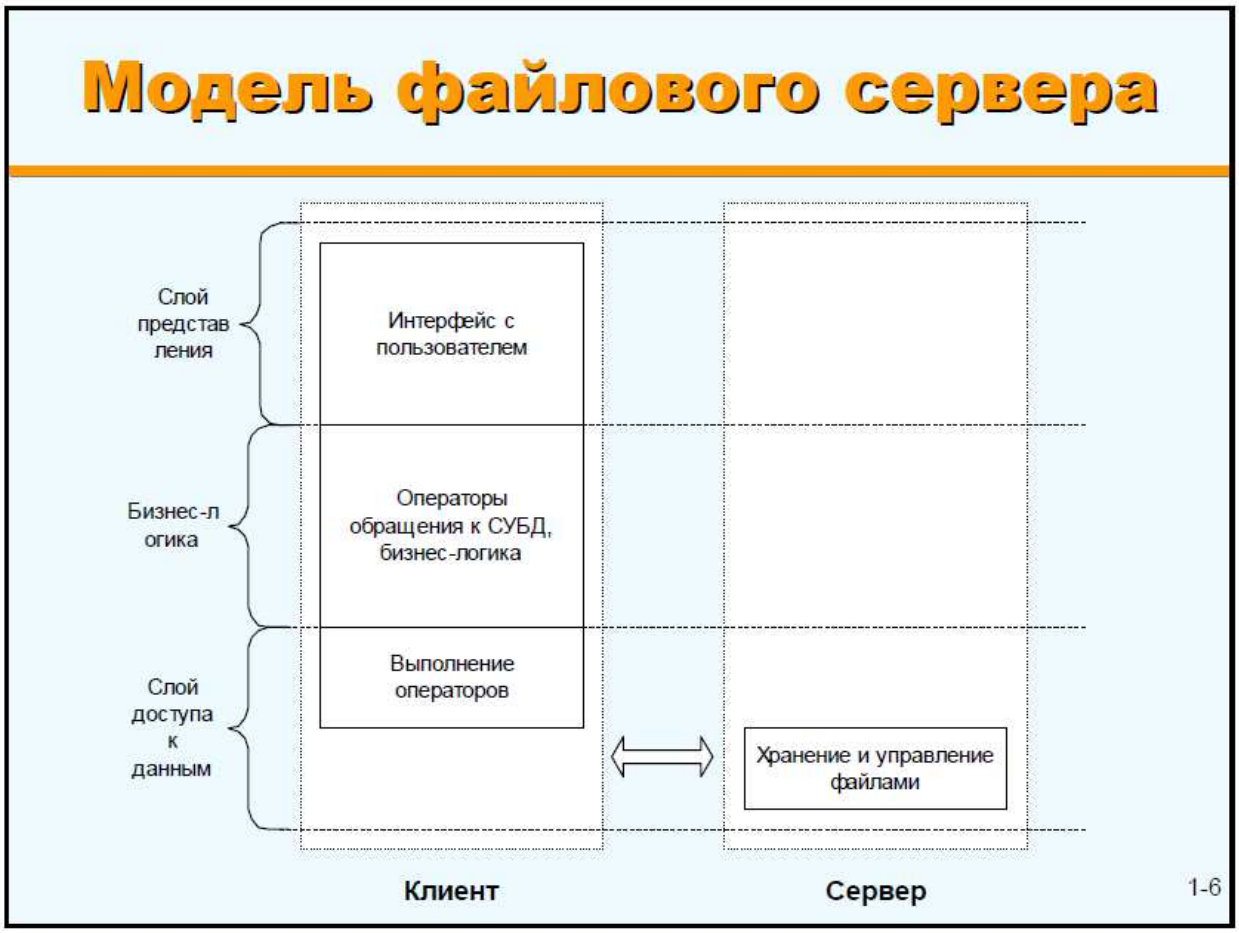
Традиционные архитектуры информационных систем.

Файл-серверная архитектура

Появились локальные сети. Файлы начали передаваться по сети. Сначала были одноранговые сети - все компьютеры равноправны.



Потом возникла идея хранения всех общедоступных файлов на выделенном компьютере в сети - файл-сервере.



Файл-серверные приложения — приложения, схожие по своей структуре с локальными приложениями и использующие сетевой ресурс для хранения программы и данных. Функции сервера: хранения данных и кода программы. Функции клиента: обработка данных происходит исключительно на стороне клиента.

Количество клиентов ограничено десятками.

Плюсы:

1. Многопользовательский режим работы с данными;
2. Удобство централизованного управления доступом;
3. Низкая стоимость разработки;

Минусы:

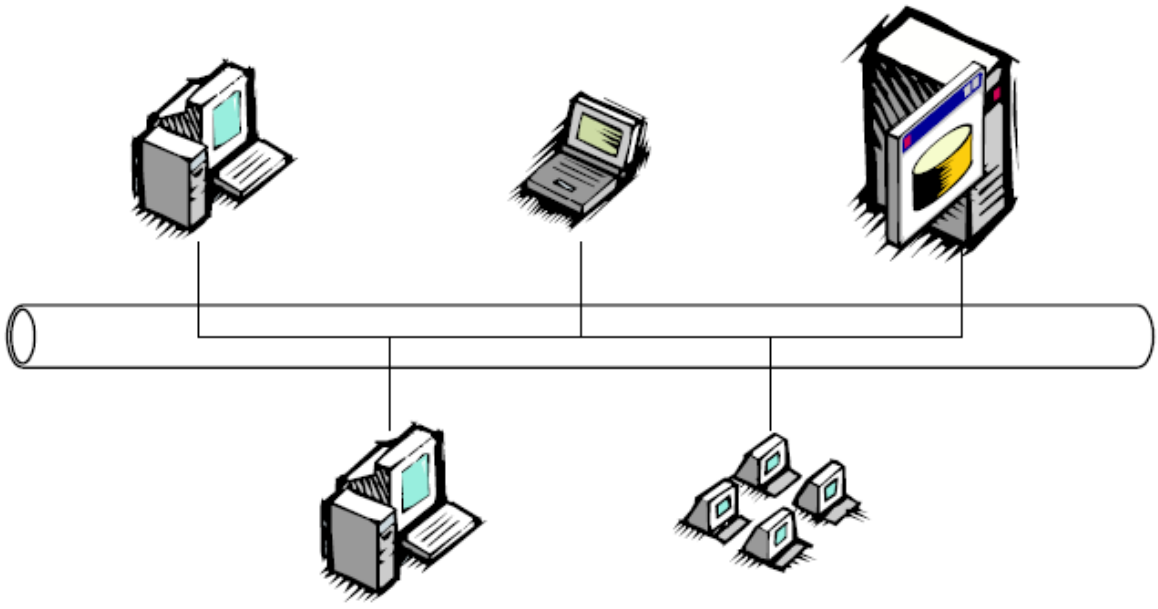
1. Низкая производительность;
2. Низкая надежность;
3. Слабые возможности расширения;

Недостатки архитектуры с файловым сервером очевидны и вытекают главным образом из того, что данные хранятся в одном месте, а обрабатываются в другом. Это означает, что их нужно передавать по сети, что приводит к очень высоким нагрузкам на сеть и, вследствие этого, резкому снижению производительности приложения при увеличении числа одновременно работающих клиентов. Вторым важным недостатком такой архитектуры является децентрализованное решение проблем целостности и согласованности данных и одновременного доступа к данным. Такое решение снижает надежность приложения.

Клиент-серверная архитектура

Ключевым отличием архитектуры клиент-сервер от архитектуры файл-сервер является абстрагирование от внутреннего представления данных (физической схемы данных). Теперь клиентские программы манипулируют данными на уровне логической схемы.

Итак, использование архитектуры клиент-сервер позволило создавать надежные (в смысле целостности данных) многопользовательские ИС с централизованной базой данных, независимые от аппаратной (а часто и программной) части сервера БД и поддерживающие графический интерфейс пользователя (ГИП) на клиентских станциях, связанных локальной сетью. Причем издержки на разработку приложений существенно сокращались.



Основные особенности:

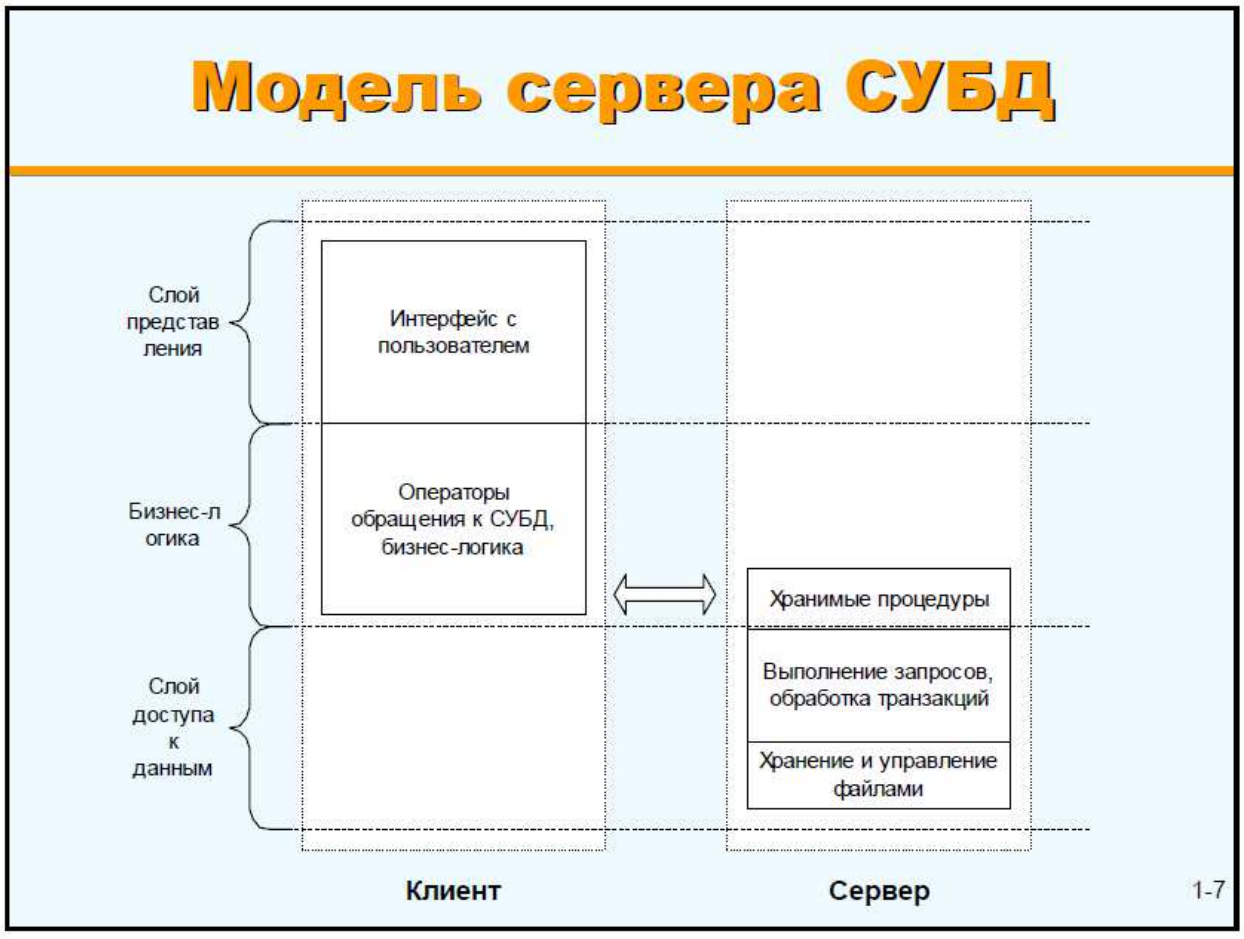
- Клиентская программа работает с данными через запросы к серверному ПО.
- Базовые функции приложения разделены между клиентом и сервером.

Плюсы:

- Полная поддержка многопользовательской работы
- Гарантия целостности данных

Минусы:

- Бизнес логика приложений осталась в клиентском ПО. При любом изменении алгоритмов, надо обновлять пользовательское ПО на каждом клиенте.
- Высокие требования к пропускной способности коммуникационных каналов с сервером, что препятствует использование клиентских станций иначе как в локальной сети.
- Слабая защита данных от взлома, в особенности от недобросовестных пользователей системы.
- Высокая сложность администрирования и настройки рабочих мест пользователей системы.
- Необходимость использовать мощные ПК на клиентских местах.
- Высокая сложность разработки системы из-за необходимости выполнять бизнес-логику и обеспечивать пользовательский интерфейс в одной программе.



Нетрудно заметить, что большинство недостатков классической или 2-х слойной архитектуры клиент-сервер проистекают от использования клиентской станции в качестве исполнителя бизнес-логики ИС. Поэтому очевидным шагом дальнейшей эволюции архитектур ИС явилась идея "тонкого клиента", то есть разбиения алгоритмов обработки данных на части связанные с выполнением бизнес-функций и связанные с отображением информации в удобном для человека представлении. При этом на клиентской машине оставляют лишь вторую часть, связанную с первичной проверкой и отображением информации, перенося всю реальную функциональность системы на серверную часть.

Переходная к трехслойной архитектуре (2.5 слоя)

Использование хранимых процедур и вычисление данных на стороне сервера сокращают трафик, увеличивают безопасность. Клиент все равно реализует часть бизнес-логики.

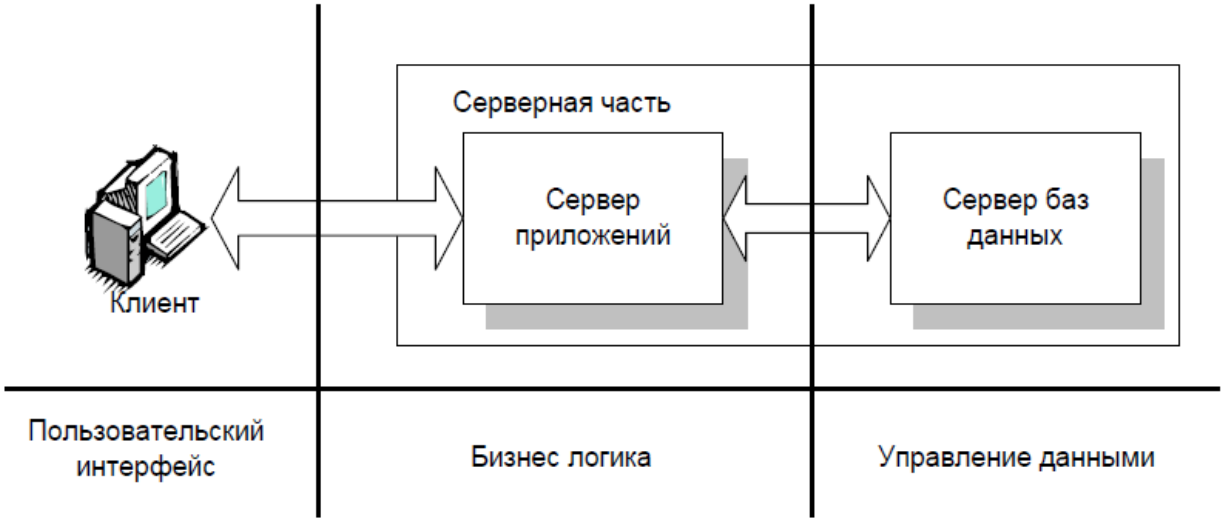
Как видно, такая организация системы весьма напоминает организацию первых унитарных систем с той лишь разницей, что на пользовательском месте стоит не терминал (с пресловутым зеленым экраном), а персональный компьютер, обеспечивающий ГИП, например, в последнее время в качестве клиентских программ часто применяют стандартные www-броузеры. Конечно, такой возврат к почти унитарным системам произошел уже на ином технологическом уровне. Обязательным стало использование СУБД со всеми их преимуществами. Программы для серверной части пишут, в основном, на специализированных языках, пользуясь механизмом хранимых процедур сервера БД. Таким образом, на уровне логической организации, ИС в архитектуре клиент-сервер с тонким клиентом расщепляется на три слоя - слой данных, слой бизнес-функций (хранимые процедуры) и слой представления. К сожалению, обычно, в такой схеме построения ИС не удается написать всю бизнес-логику приложения на не предназначенных для этого встроенных языках СУБД. Поэтому, очень часто часть бизнес-функций реализуется в клиентской части систем, которая от этого неотвратимо "толстеет". Отчасти поэтому, отчасти потому, что физически такие ИС состоят из двух компонентов, эту архитектуру часто называют 2.5-слойный клиент-сервер.

В отличие от 2-х слойной архитектуры 2.5-слойная архитектура обычно не требует наличия высокоскоростных каналов связи между клиентской и серверной частями системы, так как по сети передаются уже готовые результаты вычислений - почти все вычисления производятся на серверной стороне. Существенно улучшается также и защита информации - пользователям даются права на доступ к функциям системы, а не на доступ к ее данным и т.д. Однако вместе с преимуществами унитарного подхода архитектура 2.5 перенимает и все его недостатки, как-то: ограниченную масштабируемость, зависимость от программной платформы, ограниченное использование сетевых вычислительных ресурсов. Кроме того программы для серверной части системы пишутся на встроенных в СУБД языках описания хранимых процедур, предназначенных для валидации данных и построения несложных отчетов, а вовсе не для написания ИС масштаба предприятия. Все это снижает быстродействие системы, повышает трудоемкость создания с модификации ИС и самым негативным образом сказывается на стоимости аппаратных средств, необходимых для ее функционирования.

Трехуровневая клиент-серверная архитектура

Для решения этих проблем и была предложена так называемая 3-х слойная архитектура клиент-сервер. Основным ее отличием от архитектуры 2.5 является **физическое** разделение программ, отвечающих за хранение данных (СУБД) от программ эти данные

обрабатывающих (сервер приложения (СП), application server (AS)). Такое разделение программных компонент позволяет оптимизировать нагрузки как на сетевое, так и на вычислительное оборудование комплекса.



Компоненты трёхзвенной архитектуры, с точки зрения программного обеспечения реализуют определенные сервера БД, web-сервера и браузеры. Место любого из этих компонентов может занять программное обеспечение любого производителя. Ниже представлено описание взаимодействия компонентов трехуровневой архитектуры клиент-серверного приложения. Сервер БД представлен MySQL-сервером; сервер приложений

технологиями: ADO.NET, ASP.NET и web-сервером IIS; роль клиента выполняет любой web-браузер.

Браузер клиента *1->* **Сервер IIS** *2->* **Исполняющая среда ASP.NET 2.0** *3->* **Провайдер данных ADO.NET 2.0** *4->* **Сервер MySQL** *5->* **Провайдер данных ADO.NET 2.0** *6->* **Исполняющая среда ASP.NET 2.0** *7->* **Сервер IIS** *8->* Браузер клиента

- 1 — браузер клиента отправляет HTTP-запрос;
- 2 — на стороне сервера служба Web Internet Information Server (web-сервер IIS) определяет тип запрашиваемого ресурса, и для случая запроса *.aspx (расширение файлов страниц ASP.NET) загружает соответствующее ему (запросу) расширение Internet Server Aplication Programming Interface (ISAPI). Для страниц aspx это расширение isapi_aspnet.dll. IIS также осуществляет идентификацию и авторизацию пользователя от которого поступил запрос. В свою очередь расширение isapi_aspnet.dll загружает фабрику обработчиков ASP.NET. Далее, фабрика обработчиков создает объектную модель запрашиваемой страницы и обрабатывает действия пользователя.
- 3 — в ходе генерации ответа приложению ASP.NET может потребоваться обращение к БД, в этом случае используя библиотеки классов провайдера данных ADO.NET 2.0, выполняющая среда обращается к серверу БД;
- 4 — провайдер данных ADO.NET 2.0 передает запрос на операцию с БД серверу MySQL;
- 5 — сервер MySQL осуществляет обработку запроса, выполняя соответствующие операции с БД ;
- 6 — провайдер данных ADO.NET 2.0 передает результаты запроса объекту страницы;
- 7 — объект страницы с учетом полученных данных осуществляет рендеринг графического интерфейса страницы и направляет результаты в выходной поток;
- 8 — сервер IIS отправляет содержимое сгенерированной страницы клиентскому браузеру.

Плюсы:

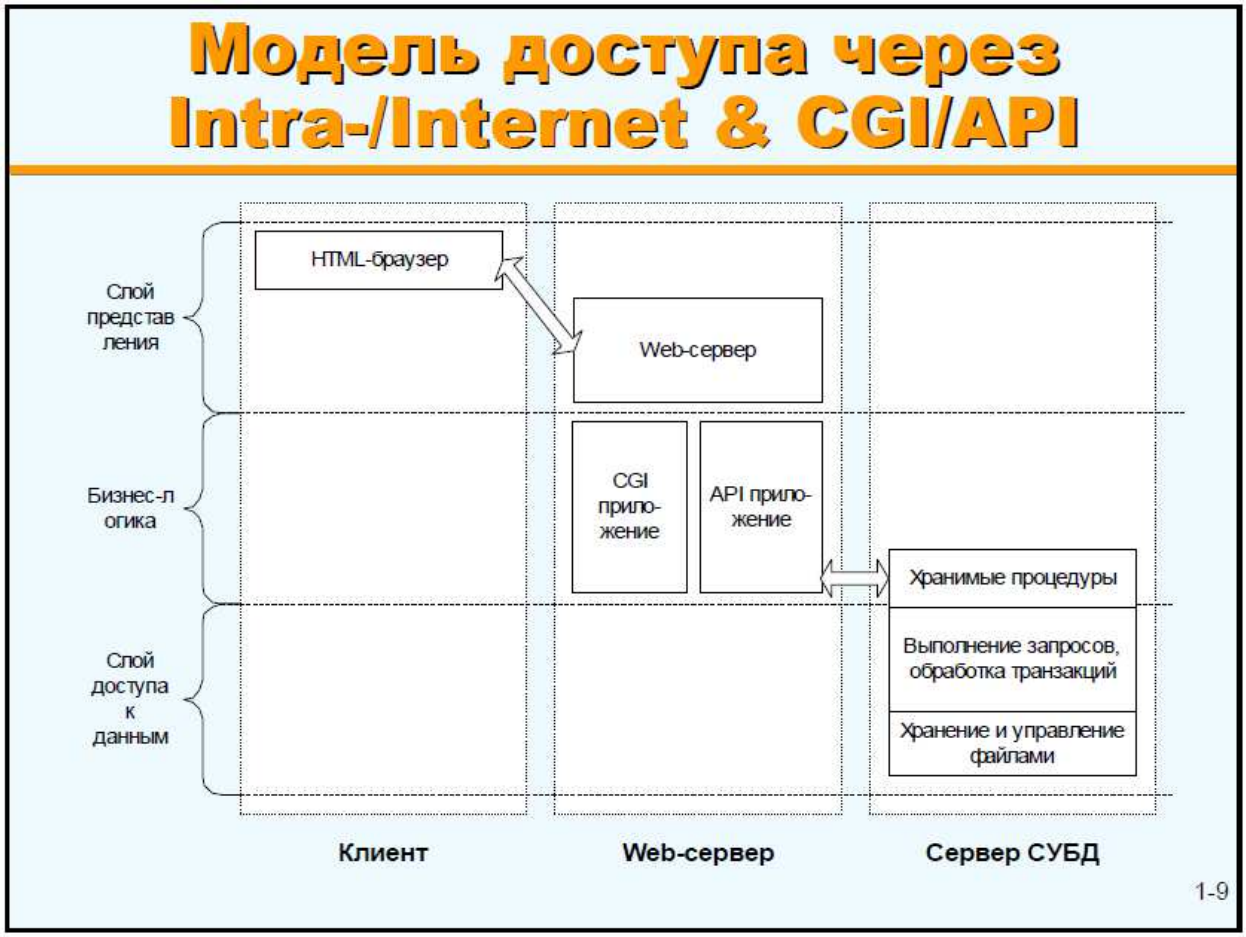
1. Тонкий клиент.
2. Между клиентской программой и сервером приложения передается лишь минимально необходимый поток данных - аргументы вызываемых функций и возвращаемые от них значения. Это теоретический предел эффективности использования линий связи, даже работа с ANSI-терминалами (не говоря уже об использование протокола http) требует большей нагрузки на сеть.
3. Сервер приложения ИС может быть запущен в одном или нескольких экземплярах на одном или нескольких компьютерах, что позволяет использовать вычислительные мощности организации столь эффективно и безопасно как этого пожелает администратор ИС.
4. Дешевый трафик между сервером приложений и СУБД. Трафик между сервером приложений и СУБД может быть большим, однако это всегда трафик локальной сети, а их пропускная способность достаточно велика и дешева. В крайнем случае, всегда можно запустить СП и СУБД на одной машине, что автоматически сведет сетевой трафик к нулю.
5. Снижение нагрузки на сервер данных по сравнению с 2.5-слойной схемой, а значит и повышение скорости работы системы в целом.
6. Дешевле наращивать функциональность и обновлять ПО.

Минусы:

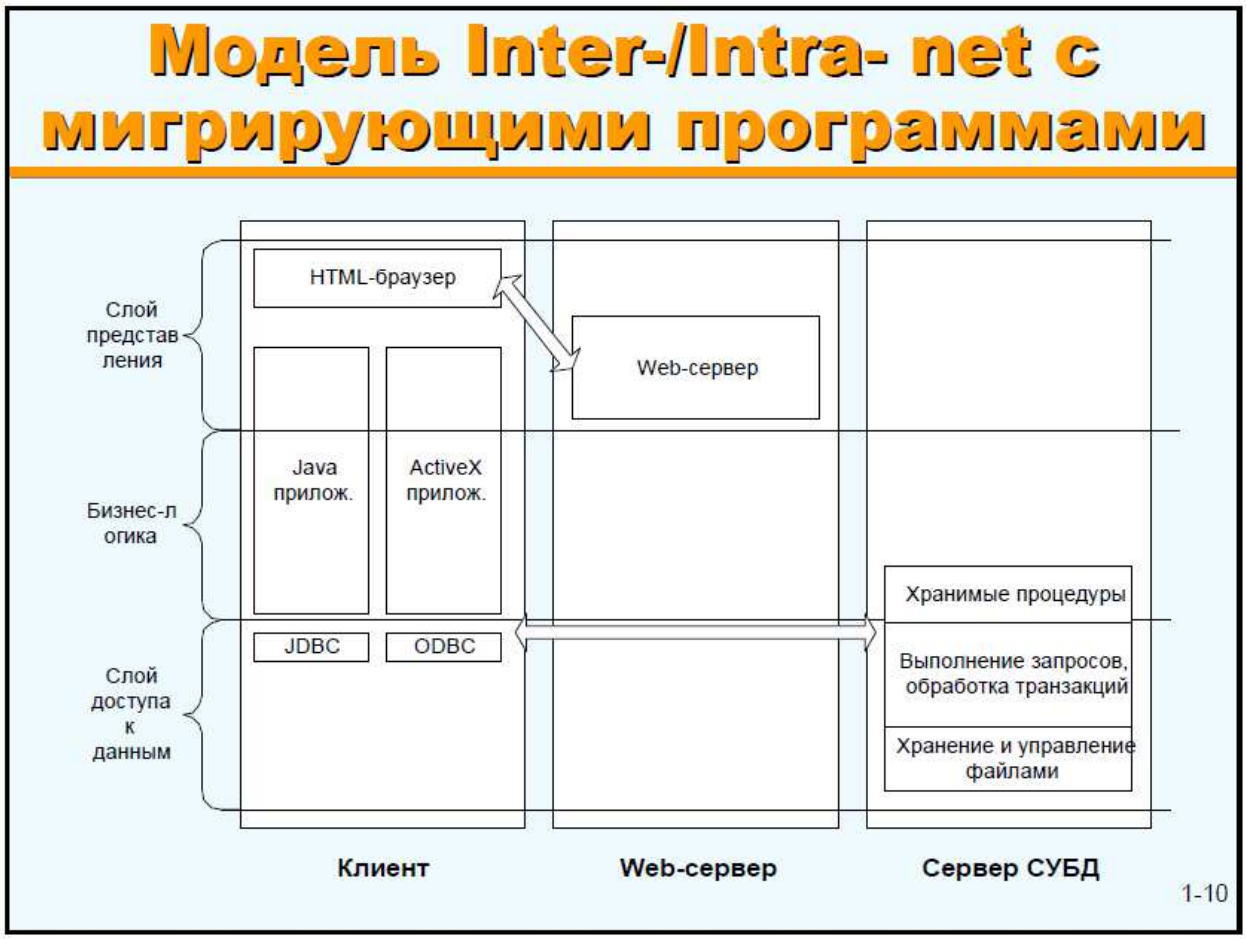
1. Выше расходы на администрирование и обслуживание серверной части.

Масштабируемость систем выполненных в 3-х слойной архитектуре впечатляет. Одна и та же система может работать как на одном отдельно стоящем компьютере, выполняя на нем программы СУБД, СП и клиентской части, так и в сети, состоящей из сотен и тысяч машин. Как уже было отмечено, единственным фактором, препятствующим бесконечной масштабируемости, является лишь требование ведения единой базы данных.

Помимо требования увеличения производительности системы с ростом масштабов деятельности важным фактором является и расширение ее функциональной наполненности. И здесь 3-х слойная схема выигрывает у своих предшественников. Для расширения функциональности не обязательно менять **всю** систему как в случае 2.5-слойной схемы - достаточно установить новый сервер приложения с требуемой функцией. Отпадают и многие проблемы связанные с переустановкой клиентских частей программы на множестве компьютеров, быть может весьма удаленных, столь актуальные для 2-слойной схемы - парадигма "тонкого" клиента предоставляет для этого целый ряд возможностей.


Модель доступа через Intra-/Internet & CGI/API
Слой представления
Бизнес-логика
Слой доступа к данным
HTML-браузер
Web-сервер
CGI приложение
API приложение
Хранимые процедуры
Выполнение запросов, обработка транзакций
Хранение и управление файлами
Клиент
Web-сервер
Сервер СУБД
1-9

Архитектура на основе Internet/Intranet с мигрирующими программами



Распределенные информационные системы.

В литературе можно найти различные определения распределенных систем, причем ни одно из них не является удовлетворительным и не согласуется с остальными.

Для наших задач хватит достаточно вольной характеристики.

Распределенная система — это набор независимых вычислительных машин, представляющийся их пользователям единой объединенной системой.

В этом определении оговариваются два момента. Первый относится к аппаратуре: все машины автономны.

Второй касается программного обеспечения: пользователи думают, что имеют дело с единой системой. Важны оба момента. Позже в этой главе мы к ним вернемся, но сначала рассмотрим некоторые базовые вопросы, касающиеся как аппаратного, так и программного обеспечения.

Характеристики распределенных систем:

1. От пользователей скрыты различия между компьютерами и способы связи между ними. То же самое относится и к внешней организации распределенных систем.

2. Пользователи и приложения единообразно работают в распределенных системах, независимо от того, где и когда происходит их взаимодействие.

Распределенные системы должны также относительно легко поддаваться расширению, или масштабированию. Эта характеристика является прямым следствием наличия независимых компьютеров, но в то же время не указывает, каким образом эти компьютеры на самом деле объединяются в единую систему.

Распределенные системы обычно существуют постоянно, однако некоторые их части могут временно выходить из строя. Пользователи и приложения не должны уведомляться о том, что части системы заменены или починены или, что добавлены новые для поддержки дополнительных пользователей.

Для того чтобы поддержать представление системы в едином виде, организация распределенных систем часто включает в себя дополнительный уровень программного обеспечения, находящийся между верхним уровнем, на котором находятся пользователи и приложения, и нижним уровнем, состоящим из операционных систем.

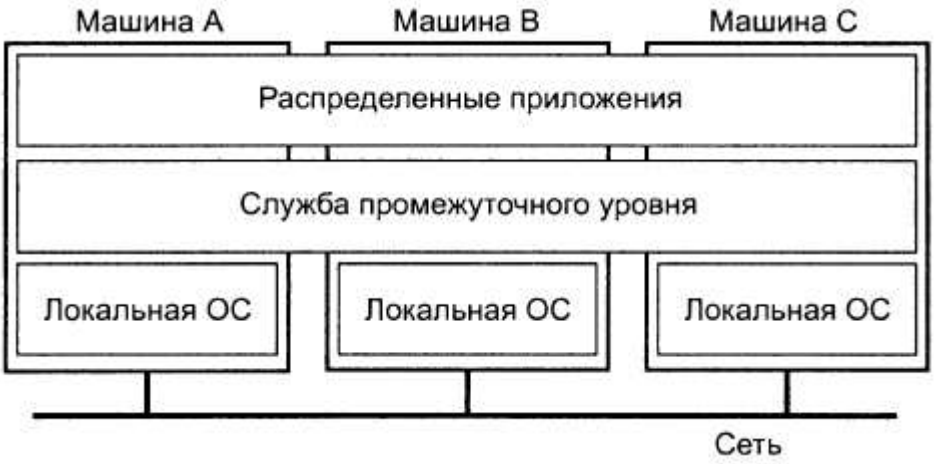


Рис. 1.1. Распределенная система организована в виде службы промежуточного уровня.

Соответственно, такая распределенная система обычно называется *системой промежуточного уровня (middleware)*. Отметим, что промежуточный уровень распределен среди множества компьютеров.

Особенности распределенных ИС

- Ссылки
- Задержки выполнения запросов
- Активация/деактивация
- Постоянное хранение
- Параллельное исполнение
- Отказы
- Безопасность

Ссылки

Ссылки на объекты в программных модулях на ОО языках программирования (например, C++) являются указателями в памяти.

1. Ссылки на объекты в распределенных системах в противоположность являются более комплексными:
   1.1.Содержат информацию о размещении
   1.2.Информацию о безопасности
   1.4.Ссылки на объектные типы
2. Ссылки на распределенные объекты значительно больше (40 байт для Orbix)

Задержки выполнения запросов

Локальные вызовы требуют порядка пары сотен наносекунд

Запрос к объекту требует от 0.1 до 10 миллисекунд

Интерфейсы в распределенной системе должны быть спроектированы так, чтобы снизить время выполнения запросов:

1. Снизить частоту обращения;
2. Укрупнить выполняемые функции.

Активация/Деактивация

Объекты в ОО языках находятся в виртуальной памяти от создания до уничтожения

В распределенных системах

1. Больше объектов
2. Объекты могут не использоваться на протяжении долгого времени

Реализации распределенных объектов

1. Переносятся в память при активации
2. Удаляются из памяти при деактивации

Постоянное хранение

Объекты могут иметь или не иметь состояние.

Объекты имеющие состояние должны сохранять его на постоянный носитель между:

1. Деактивацией объекта
2. Активацией объекта

Может быть достигнуто:

1. Записью в файловую систему
2. Отражением на реляционные БД
3. С помощью объектных БД

Параллельное исполнение

В нераспределенных системах исполнение в основном последовательное, иногда конкурентное в разных нитях процессов.

Распределенные компоненты выполняются параллельно, что приводит к необходимости согласования выполнения.

Отказы

Запросы в распределенных системах имеют большую вероятность отказов

Клиенты обязаны проверять факт выполнения запросов сервером

Безопасность

Безопасность в ОО приложениях может выполняться на основе контроля сеансов.

При работе распределенных систем возникают вопросы безопасности:

1. Кто запрашивает выполнение операции?
2. Как мы можем удостовериться, что субъект является именно тем за кого он себя выдает?
3. Как мы примем решение предоставлять или нет субъекту право на выполнение сервиса?
4. Как мы можем неопровержимо доказать, что сервис был предоставлен?

Источники

1. Бурдаков А.В. – Распределенные вычислительные системы. Лекция №1, слайд 3-5.
2. Кузнецов – Архитектуры ИС. Слайд 12, 14, 20,21,22,23, 24, 25;
3. СофтСиб – Преимущества клиент-серверной перед файл-серверной технологией. http://www.soft-sib.ru/articles/programs/10/
4. Танненбаум – Распределенные вычислительные системы. Глава 1.1
5. Бурдаков А.В. – Распределенные вычислительные системы. Лекция №3, слайд 3-5.

Лекция 4. Объектно-ориентированное проектирование, Этапы проектирования ИС.

Аннотация: Объектно-ориентированное проектирование. Этапы проектирования ИС.

1. Введение

От структур и подпрограмм к объектам.

Целями **структурного программирования** являлись структуризация данных и декомпозиция кода. Объединение данных в структуры позволяло, с одной стороны, более естественно описывать сущности реального мира, имеющие самые разнотипные наборы атрибутов. А, с другой стороны, делало процесс разработки программ более простым, поскольку однородные данные были сгруппированы в одном месте и под одним общим именем.

Декомпозиция кода заключалась в разделении исходного кода программы на отдельные подпрограммы. Подпрограммы снижали количество возможных связей между отдельными операторами (локализация кода) и, кроме того, позволяли устанавливать необходимую область видимости для переменных, используемых в подпрограммах (локализация данных). Локализация кода не позволяла произвольно переходить от одного оператора в одной подпрограмме к любому оператору в другой подпрограмме. Вызов подпрограммы приводил к передаче управления только в определенную точку входа, хотя некоторые языки допускали более одной точки входа в подпрограмму.

Такое решение не только повышало надежность программ, но и позволяло многократно использовать одни и те же подпрограммы в различных программах. Как следствие, стали появляться библиотеки подпрограмм, как самостоятельные программные продукты. Часть библиотек поставлялась вместе с компиляторами с языков программирования и операционными системами, другая часть распространялась свободно или на коммерческой основе. Применение библиотек существенно ускоряло процесс разработки программного обеспечения, так как подпрограммы, применяемые многократно, требовалось отлаживать только единожды.

Локализация данных имела ничуть не меньшее значение, чем локализация кода. Три вида данных в подпрограмме:

- Локальные данные подпрограммы, которые автоматически создавались при ее вызове и уничтожались при возврате из подпрограммы;
- Локальные данные, сохраняемые между несколькими вызовами одной подпрограммы;
- Глобальные данные, доступные нескольким или всем подпрограммам, и доступ к которым осуществлялся прямо из подпрограммы;
- Фактические параметры, передаваемые подпрограммам, замещающие объявленные формальные параметры.

Глобальные данные применялись в основном для организации связей между подпрограммами, но они существенно ограничивали гибкость подпрограмм. Действительно, если подпрограмма работала с какими-то глобальными данными, то эти данные должны были совпадать по имени, типу во всех программах, которые использовали данную подпрограмму.

В отличие от этого, локальные данные, наоборот, позволяли подпрограмме быть независимой от программ, использующих ее. Но они не обеспечивали передачу данных между различными подпрограммами. Для передачи данных или ссылок на них использовался механизм формальных/фактических параметров. При описании подпрограммы объявляются формальные параметры, которые она принимает при вызове. Когда реально происходит вызов подпрограммы, на место формальных параметров подставляются фактические параметры того же типа.

Образование библиотек подпрограмм происходило, как правило, по функциональному признаку. То есть, в одну библиотеку помещались подпрограммы, выполняющие близкую по составу работу. Например, библиотека подпрограмм, работающих с каким-то устройством. Библиотеки имели существенный недостаток. Результаты их работы могли быть неверны при подаче на вход не корректных данных. Однако данные располагались в программе за пределами библиотеки и могли меняться произвольно. Появилась реальная потребность объединить данные и код, который их обрабатывает. Эта задача была решена в **модульном программировании**. Модуль, в отличие от библиотеки подпрограмм, мог содержать не только подпрограммы, но и данные.

Данные, располагаемые в модуле, были глобальными и для модуля и для программы, использующей модуль. Для увеличения безопасности данных модуль разделили на несколько зон. Одна из зон обеспечивала взаимодействие между модулем и внешним программные обеспечением. Это так называемая зона интерфейса. Другая зона отвечала за реализацию модуля и была недоступна для внешнего программного обеспечения.

Соответственно, критичные к произвольным изменениям данные помещались в зону реализации и были недоступны для программы и других модулей. Про эти данные можно сказать, что они инкапсулированы модулем.

В интерфейсной зоне оставались, как правило, только те данные, с которыми активно взаимодействовало внешнее программное обеспечение. Нельзя сказать, что эти данные можно было произвольно изменять без ущерба для работоспособности программы. Практически все данные были инкапсулированы, то есть помещены в зону реализации, а для взаимодействия с ними в интерфейсную зону помещались объявления специальных интерфейсных программ. Только они могли изменять значения переменных внутри модуля или возвращать значения этих переменных.

Перенос данных в область реализации поставил еще одну проблему: данные необходимо было инициализировать перед началом работы и освобождать перед окончанием работы модуля. Например, перед началом работы модуля могла потребоваться динамически распределяемая область памяти, а в конце требовалось вернуть эту область памяти системе. Это привело к появлению в модулях областей инициализации, которые вызывались до начала работы модуля, и областей, которые автоматически вызывались перед тем, как модуль завершит работу.

Если теперь сложить все элементы полученной мозаики воедино, то получится знакомая картина: все подпрограммы модуля собраны вместе по функциональному признаку и работают над одними данными; модуль имеет области кода, которые вызываются при инициализации и окончании работы модуля. Такой модуль можно назвать прообразом **объекта**.

Мы вплотную подошли к возможности реализации объектной технологии. Фактически осталось только ввести поддержку полиморфизма и наследования, что сегодня, как правило, реализуется в рамках компиляторов. Важно понимать, что объектная технология, являясь прямым продолжением технологии структурной разработки программ, открыла новые принципиально иные подходы к проектированию сложного программного обеспечения.

Объектно-ориентированное программирование

Абстракция

Абстрагирование — это способ выделить набор значимых характеристик объекта, исключая из рассмотрения незначимые. Соответственно, абстракция — это набор всех таких характеристик.

Инкапсуляция

Инкапсуляция — это свойство системы, позволяющее объединить данные и методы, работающие с ними, в классе и скрыть детали реализации от пользователя.

Класс

Класс является описываемой на языке терминологии (пространства имён) исходного кода моделью ещё не существующей сущности (объекта). Фактически он описывает устройство объекта, являясь своего рода чертежом. Говорят, что объект — это экземпляр класса. При этом в некоторых исполняющих системах класс также может представляться некоторым объектом при выполнении программы посредством динамической идентификации типа данных. Обычно классы разрабатывают таким образом, чтобы их объекты соответствовали объектам предметной области.

Наследование

Наследование — это свойство системы, позволяющее описать новый класс на основе уже существующего с частично или полностью заимствующейся функциональностью. Класс, от которого производится наследование, называется базовым, родительским или суперклассом. Новый класс — потомком, наследником или производным классом.

Объект

Сущность в адресном пространстве вычислительной системы, появляющаяся при создании экземпляра класса или копирования прототипа (например, после запуска результатов компиляции и связывания исходного кода на выполнение).

Полиморфизм

Полиморфизм — это свойство системы использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта.

Прототип

Прототип — это объект-образец, по образу и подобию которого создаются другие объекты. Объекты-копии могут сохранять связь с родительским объектом, автоматически наследуя изменения в прототипе; эта особенность определяется в рамках конкретного языка.

Аспектно-ориентированное программирование

Аспектно-ориентированное программи́рование (АОП) — парадигма программирования, основанная на идее разделения функциональности для улучшения разбиения программы на модули.

Основные понятия АОП:

Аспект — модуль или класс, реализующий сквозную функциональность. Аспект изменяет поведение остального кода, применяя совет в точках соединения, определённых некоторым срезом.

Совет — средство оформления кода, который должен быть вызван из точки соединения. Совет может быть выполнен до, после или вместо точки соединения.

Точка соединения — точка в выполняемой программе, где следует применить совет. Многие реализации АОП позволяют использовать вызовы методов и обращения к полям объекта в качестве точек соединения.

Срез — набор точек соединения. Срез определяет, подходит ли данная точка соединения к данному совету. Самые удобные реализации АОП используют для определения срезов синтаксис основного языка (например, в AspectJ применяются Java-сигнатуры) и позволяют их повторное использование с помощью переименования и комбинирования.

Внедрение — изменение структуры класса и/или изменение иерархии наследования для добавления функциональности аспекта в инородный код. Обычно реализуется с помощью некоторого метаобъектного протокола (англ. metaobject protocol, MOP).

Концептуальные положения объектных моделей

При рассмотрении объектных моделей обычно выделяют **три основных концептуальных положения: инкапсуляция, полиморфизм и наследование**. Описание концептуальных положений создает представление о том, какие возможности заложены в данную технологию, на какие горизонты эта технология может вывести процесс разработки программного обеспечения.

Сложные системы.

Объектная технология открывает возможность построения гораздо более сложных систем и программных комплексов, чем допускала технология структурного программирования. Разница в подходах к разработке принципиальна и она базируется именно на высокой сложности современного программного обеспечения. Как результат применение структурной технологии ведет к слишком продолжительному циклу разработки, сложности в модернизации и управлении, сокращенному жизненному циклу и повышенной стоимости.

Гради Буч в своей книге «Объектно-ориентированный анализ и проектирование с примерами приложений на C++» со ссылкой на Брукса отмечает, что сложность вызывается четырьмя основными причинами:

1. сложностью реальной предметной области,
2. трудностью управления процессом разработки,
3. необходимостью обеспечить достаточную гибкость программы,
4. неудовлетворительными способами описания поведения больших дискретных систем.

Далее Буч выделяет пять признаков сложных систем:

- Сложные системы часто являются иерархичными и состоят из связанных подсистем.
- Выбор, какие компоненты в данной системе считаются элементарными, как правило на усмотрение исследователя.
- Внутрикомпонентная связь обычно сильнее, чем связь между компонентами.
- Иерархические системы обычно состоят из немногих типов подсистем, по-разному скомбинированных и организованных.

- Любая работающая сложная система является результатом развития работавшей более простой системы.

Сложная система, как было отмечено ранее, отличается сложностью управления, то есть сложностью связей между компонентами. Каждый уровень управления обладает собственной логикой, которая не зависит от логики управления других уровней. Взаимодействие между уровнями управления в любой сложной системе основано на декларированном интерфейсе каждого уровня. Создание интерфейсов и связей на каждом уровне управления представляет собой отдельную задачу и требует отдельных проектных решений. Создание и развитие каждого уровня может и должно вестись параллельно с созданием и развитием других уровней. Развитие любого уровня начинается с декларации его интерфейса и только после определения интерфейса и увязки его с интерфейсами смежных уровней управления, можно переходить к реализации всех уровней. Интерфейс представляет собой формальную спецификацию возможностей данного логического уровня. Этот тезис переформулирует положение о трудностях, связанных с процессом разработки. Трудность разработки зависит от того, насколько точно определены основные логические уровни и того, насколько хорошо формализованы интерфейсы всех уровней.

Умение выполнять декомпозицию системы адекватно умению превращать сложную систему в простую. Так как система находится в постоянном развитии, здесь не применимы методы, используемые для разработки программы как законченного продукта. Развитие системы должно происходить непрерывно от экспериментальной модели до моделей, находящихся в эксплуатации.

2. Принципы ООП

2.1. Класс, объект

Любая сущность — объект

В настоящем объектно-ориентированном языке все элементы так называемой *предметной области* (problem domain) выражаются через концепцию *объектов.* Как вы уже, наверное, поняли, объекты — это центральная идея объектно-ориентированного программирования. Многие из нас, обдумывая какую-то проблему, вряд ли оперируют понятиями "структура", "пакет данных", "вызов функций" и "указатели", ведь привычнее применять понятие "объекты". Возьмем такой пример.

Допустим, вы создаете приложение для выписки счета-фактуры, в котором нужно подсчитать сумму по всем позициям. Какая из двух формулировок понятней с точки зрения пользователя?

- **Не объектно-ориентированный подход** Заголовок счета-фактуры представляет структуру данных, к которой я получу доступ. В эту структуру войдет также дважды связанный список структур, содержащих описание и стоимость каждой позиции. Поэтому для получения общего итога по счету мне потребуется объявить переменную с именем наподобие *totallnvoiceAmount* и инициализировать ее нулем, получить указатель на головную структуру счета, получить указатель на начало связанного списка, а затем "пробежать" по всему этому списку. Просматривая структуру для каждой позиции, я буду брать оттуда переменную-член, где находится итог для данной позиции, и прибавлять его к *totallnvoiceAmount.*
- **Объектно-ориентированный подход** У меня будет объект "счет-фактура", и ему я отправлю сообщение с запросом на получение общей суммы. Мне не важно, как информация хранится внутри объекта, как это было в предыдущем случае. Я

общаюсь с объектом естественным образом, запрашивая у него информацию посредством сообщений. (Группа сообщений, которую объект в состоянии обработать, называется *интерфейсом* объекта. Чуть ниже я объясню, почему в объектно-ориентированном подходе вместо термина "реализация" правильней употреблять термин "интерфейс".)

Очевидно, что объектно-ориентированный подход естественнее и ближе к тому способу рассуждений, которым многие из нас руководствуются при решении задач. Во втором варианте объект "счет-фактура", наверно, просматривает в цикле *совокупность*(collection) объектов, представляющих данные по каждой позиции, посылая им запросы на получение суммы по данной позиции. Но если требуется получить только общий итог, то *вам все равно, как это реализовано,* так как одним из основных принципов объектно-ориентированного программирования является *инкапсуляция* (encapsulation). Инкапсуляция — это свойство объекта скрывать свои внутренние данные и методы, представляя наружу только интерфейс, через который осуществляется программный доступ к самым важным элементам объекта. Как объект выполняет задачу, не имеет значения, главное, чтобы он справлялся со своей работой. Имея в своем распоряжении интерфейс объекта, вы заставляете объект выполнять нужную вам работу. (Ниже я остановлюсь на понятиях "инкапсуляция" и "интерфейс".) Здесь важно отметить, что разработка и написание программ моделирования реальных объектов предметной области облегчается тем, что представить поведение таких объектов довольно просто.

Заметьте: во втором подходе от объекта требовалось, чтобы он произвел нужную вам работу, т. е. подсчитал общий итог. В отличие от структуры, в объект по определению входят не только данные, но и методы их обработки. Это значит, что при работе с некоторой проблемной областью можно не только создать нужные структуры данных, но и решить, какие методы связать с данным объектом, чтобы объект стал полностью инкапсулированной частью функциональности системы.

2.2. Инкапсуляция

Инкапсуляция подразумевает объединение и защиту кода и данных в некоторой сущности: объекте или модуле. Для чего вообще нужна инкапсуляция? Для того, чтобы защитить реализацию класса. Если не обеспечивается должная защита, то разработчик, использующий данный класс, имеет возможность рассчитывать на определенную реализацию. В этом случае теряется возможность простой модификации не только данного класса, но и всей системы в целом. Предположим, что существует возможность обращаться к любому оператору подпрограммы из внешнего программного обеспечения. Это нарушает инкапсуляцию кода программы, но одновременно утрачивается и возможность модификации самой подпрограммы. Нельзя изменить реализацию по той причине, что какое-то внешнее ПО могло обращаться к одному из изменяемых операторов.

Кроме того, инкапсуляция прямо связана с требованием повышения надежности систем, иначе объекты могут произвольно менять данные других объектов.

Третьим достоинством инкапсуляции является упрощенное восприятие объекта его пользователем. Пользователь видит только те свойства объекта, которые сможет использовать, абстрагируясь от сложности реализации.

2.3. Наследование

Наследование это передача подклассу свойств, структуры и поведения суперкласса. Однако передача свойств и структуры это механизм, а не цель. Реальных целей несколько. К ним относится, например, упорядочивание типов сущностей, то есть классификация сущностей в системе. Действительно, не так просто разобраться в ньюансах, отличающих один тип сущности от другого типа, если не представлять иерархию их развития, ее идеологическую основу.

Благодаря классификации становится возможным введение и широкое практическое использование абстракции. Абстракция дает возможность исключить из рассмотрения малозначительные детали и сосредоточиться на том, что действительно важно на данном уровне рассмотрения. Как правило, абстракция используется для определения интерфейса, который позже реализуется и детализируется в подклассах. Например, можно реализовать абстрактное предприятие и определить необходимый интерфейс, который позволит взаимодействовать предприятию с внешним миром: другими предприятиями, государственными учреждениями и т.п. При этом неважно, чем будет заниматься это предприятие.

Абстракцию также полезно использовать для задания поведения класса. Здесь под поведением понимается последовательность действий по обработке какого-то сообщения (или реакция класса на некоторое воздействие). Подклассы наследуют реализацию данного алгоритма, что приводит не только к значительной экономии кода за счет многократного использования, но и позволяет при необходимости менять поведение множества подклассов простой заменой алгоритма суперкласса.

Итак, наследование позволяет классифицировать используемые программные сущности, вводить высокоуровневые абстракции, передавать свойства и структуру суперкласса своим подклассам, что позволяет существенно экономить код. Наконец, наследование определяет направление развития классов, что существенно облегчает разработку иерархии классов, отражающей частный, но важный случай взаимосвязи между сущностями реальной предметной области.

Полиморфизм подразумевает много форм реализации. Полиморфизм имеет большое значение для объектной технологии, так как он позволяет устанавливать аналогии между методами различных классов.

Например, для классов фигур можно использовать методы Скрыть(Hide), Отобразить (Draw). Несмотря на то, что все фигуры различны, данный шаблон поведения будет корректно работать для любой из этих фигур благодаря полиморфизму.

Обычно полиморфизм рассматривают как составную часть механизма наследования. Считается, что полиморфными могут быть только свойства (методы) подклассов. Но это узкая трактовка. Например, летать могут птицы, насекомые, механические аппараты. Означает ли это, что они имеют общего летающего предка? Конечно, нет.

Основное достоинство полиморфизма заключается в том, что он позволяет различным классам иметь семантически однородные свойства. Благодаря этому становится возможным однотипно взаимодействовать с различными классами.

Полиморфизм реализуется через наследование и интерфейсы.

3. Критика ООП

- Более низкое быстродействие.
- Не панацея от всех бед – сложность предметной области остается.

- Лучше моделирует сущности, хуже моделирует процессы, отсюда распространение функционального программирования.

4. Этапы проектирования ИС

К проектированию ИС непосредственное отношение имеют два направления деятельности:

1) проектирование ИС конкретных организаций на базе готовых программных и аппаратных компонентов;

2) проектирование компонентов ИС и инструментальных средств, ориентированных на многократное применение при разработке многих конкретных автоматизированных систем.

Сущность первого направления можно охарактеризовать словами *«системная интеграция»* (другое близкое понятие имеет название *консалтинг).* Разработчик ИС должен быть специалистом в области системотехники, хорошо знать соответствующие международные стандарты, состояние и тенденции развития информационных технологий и программных продуктов, владеть инструментальными средствами разработки приложений (CASE-средствами) и быть готовым к восприятию и анализу автоматизируемых процессов в сотрудничестве со специалистами-прикладниками.

Существует ряд фирм, специализирующихся на разработке проектов ИС (например, Price Waterhouse, ЛАНИТ, LUXOFT и др.)

Второе направление в большей мере относится к области разработки математического и программного обеспечения для реализации функций ИС - моделей, методов, алгоритмов, программ на базе знания системотехники, методов анализа и синтеза проектных решений, технологий программирования, операционных систем и т.п. Существует ряд общеизвестных технологий (методик) проектирования ПО ИС, среди которых, прежде всего, следует назвать компонентно-ориентированную разработку - технологию индустриальной разработки программных систем.

В России действует государственный стандарт на стадии создания автоматизированных систем ГОСТ 34.601-90. Существует и международный стандарт на стадии жизненного цикла программной продукции (ISO 12207:1995). Как собственно ИС, так и компоненты ИС являются сложными системами, и при их проектировании нужно использовать один из стилей проектирования:

- *нисходящее (Top-of-Design),* четкая реализация нисходящего проектирования приводит к *спиральной модели* разработки ПО, на каждом витке спирали блоки предыдущего уровня детализируются, используются обратные связи (альтернативой является так называемая *каскадная модель,* относящаяся к поочередной реализации частей системы);
- *восходящее (Bottom-of-Design);*
- *эволюционное (Middle-of-Design).*

Чаще всего применяют нисходящий стиль блочно-иерархического проектирования.

Рассмотрим этапы нисходящего проектирования ИС.

Верхний уровень проектирования ИС часто называют *концептуальным* проектированием. Концептуальное проектирование выполняют в процессе предпроектных исследований, формулировки ТЗ, разработки эскизного проекта.

Предпроектные исследования проводят путем анализа деятельности предприятия (компании, учреждения, офиса), на котором создает или модернизируется ИС. При этом нужно получить, ответы на вопросы:

1. Что устраивает в существующей технологии?
2. Что можно улучшить?
3. Кому это нужно и, следовательно, каков будет эффект?

Перед обследованием формируются и в процессе его проведения уточняются цели обследования - определение возможностей и ресурсов для повышения эффективности функционирования предприятия на основе автоматизации процессов управления, проектирования, документооборота и т.п. Содержание обследования - выявление структуры предприятия, выполняемых функций, информационных потоков, имеющихся опыта *и* средств автоматизации. Обследование проводят системные аналитики (системные интеграторы) совместно с представителями организации-заказчика.

На основе анализа результатов обследования строят модель, отражающую деятельность предприятия на данный момент (до реорганизации). Такую модель называют *«As Is (как есть).* Далее разрабатывают исходную концепцию ИС. Эта концепция включает в себя предложения по изменению структуры предприятия, взаимодействию подразделений, информационным потокам, что выражается в модели «To Be» (Как должно быть).

Результаты анализа конкретизируются в ТЗ на создание ИС. В ТЗ указывают потоки входной информации, типы выходных документов и предоставляемых услуг, уровень защиты информации, требования к производительности (пропускной способности) и т.п. ТЗ направляют заказчику для обсуждения и окончательного согласования.

Эскизный проект (техническое предложение) предоставляют в виде проектной документации, описывающей архитектуру системы, структуру ее подсистем, состав модулей. Здесь же содержатся предложения по выбору базовых программно-аппаратных средств, которые должны учитывать прогноз развития предприятия.

В отношении аппаратных средств и особенно ПО такой выбор чаще всего есть выбор фирмы-поставщика необходимых средств (или, по крайней мере, базового ПО), так как правильная совместная работа программ разных фирм достигается с большим трудом. В проекте может быть предложено несколько вариантов выбора. При анализе выясняются возможности покрытия автоматизи\руемых функций имеющимися программными продуктами и, следовательно, объемы работ по разработке оригинального ПО. Подобный анализ необходим для предварительной оценки временных и материальных затрат на автоматизацию. Учет ресурсных ограничений позволяет уточнить достижимые масштабы автоматизации, подразделить проектирование ИС на работы первой, второй очереди и т. д.

После принятия эскизного проекта разрабатывают *прототип* ИС, представляющий собой набор программ, эмулирующих работу готовой системы. Благодаря прототипированию можно не только разработчикам, но и будущим пользователям ИС увидеть контуры и особенности системы и, следовательно, заблаговременно внести коррективы в проект.

Как на этапе предпроектных исследований, так и на последующих этапах целесообразно придерживаться определенной дисциплины фиксации и представления получаемых результатов, основанной на той или иной методике формализации спецификаций. Формализация нужна для однозначного понимания исполнителями и заказчиком требований, ограничений и принимаемых решений.

При концептуальном проектировании применяют ряд спецификаций, среди которых центральное место занимают модели преобразования, хранения и передачи информации. Модели, полученные в процессе обследования предприятия, являются моделями его функционирования. В процессе разработки ИС модели, как правило, претерпевают существенные изменения (переход от «As Is» к «To Be») и в окончательном виде модель *«To Be»* рассматривают *в* качестве модели проектируемой ИС.

Различают функциональные, информационные, поведенческие и структурные модели.

- **Функциональная модель** системы описывает совокупность выполняемых системой функций.
- ***Информационная*** **модель** отражает структуры данных – их состав и взаимосвязи.
- **Поведенческая модель** описывает информационные процессы (динамику функционирования), в ней фигурируют такие категории, как состояние системы, событие, переход из одного состояния в другое, условия перехода, последовательность событий, осуществляется привязка ко времени.
- ***Структурная*** **модель** характеризует построение системы – состав подсистем, их взаимосвязи.

Содержанием последующих этапов нисходящего проектирования (согласно ГОСТ *34.601-90,* это стадия разработки технического проекта, рабочей документации, ввода в действие) являются уточнение перечней приобретаемого оборудования и готовых программных продуктов, построение системной среды, детальное инфологическое проектирование баз данных и их первоначальное наполнение, разработка собственного оригинального ПО, которая, в свою очередь, делится на ряд этапов нисходящего проектирования. Эти работы составляют содержание *рабочего проектирования.* После этого следуют закупка и инсталляция программно-аппаратных средств, внедрение и опытная эксплуатация системы.

Особое место в ряду проектных задач занимает разработка проекта корпоративной вычислительной сети, поскольку ТО ИС имеет сетевую структуру. Если территориально ИС располагается в одном здании или в нескольких близко расположенных зданиях, то корпоративная сеть может быть выполнена в виде совокупности нескольких локальных подсетей, связанных опорной локальной сетью. Кроме выбора типов подсетей, связных протоколов и коммутационного оборудования приходится решать задачи распределения узлов по подсетям, выделения серверов, выбора сетевого ПО, определения способа управления данными в выбранной схеме распределенных вычислений и т. п.

В случае, если ИС располагается в удаленных друг от друга пунктах, в частности, расположенных в разных городах, то прежде всего рассматривается возможность использования услуг Internet. Возникающие при этом проблемы связаны с обеспечением информационной безопасности и надежности доставки сообщений.

Источники

1. А.С. Усов – Письмо 1. Стр. 1 - 3.
2. А.С. Усов – Письмо 2. Стр. 4 – 8.
3. Норенков – Основы автоматизированного проектирования. Стр. 33-36
4. Веб-библиотека. Статья «Любая сущность - объект»

   http://www.weblibrary.biz/c-sharp/princypy/obekt

Лекция 5. Модели жизненного цикла ПО, проектирование как конструирование.

Аннотация.

Стандарты жизненного цикла ПО

Информационные системы должны удовлетворять интересам бизнеса, а также быть легко модифицируемыми и недорогими. Плохо спроектированная система, в конечном счете, требует больших затрат и времени для ее содержания и обновления.

Одним из базовых понятий методологии проектирования ИС является понятие жизненного цикла ее программного обеспечения (ЖЦ ПО).

Жизненный цикл ПО – это непрерывный процесс, который начинается с момента принятия решения о необходимости его создания и заканчивается в момент его полного изъятия из эксплуатации.

Существует целый ряд стандартов, регламентирующих ЖЦ ПО, а в некоторых случаях и процессы разработки.

Среди наиболее известных стандартов можно выделить следующие:

- ***ГОСТ 34.601-90*** - распространяется на автоматизированные системы и устанавливает стадии и этапы их создания. Кроме того, в стандарте содержится описание содержания работ на каждом этапе. Стадии и этапы работы, закрепленные в стандарте, в большей степени соответствуют каскадной модели жизненного цикла.

- ***ISO/IEC 12207:1995*** - стандарт на процессы и организацию жизненного цикла. Распространяется на все виды заказного ПО. Стандарт не содержит описания фаз, стадий и этапов.

- ***Custom Development Method*** (методика Oracle) по разработке прикладных информационных систем - технологический материал, детализированный до уровня заготовок проектных документов, рассчитанных на использование в проектах с применением Oracle. Применяется CDM для классической модели ЖЦ (предусмотрены все работы/задачи и этапы), а также для технологий "быстрой разработки" (Fast Track) или "облегченного подхода", рекомендуемых в случае малых проектов.

- ***Rational Unified Process (RUP)*** предлагает итеративную модель разработки, включающую четыре фазы: начало, исследование, построение и внедрение. Каждая фаза может быть разбита на этапы (итерации), в результате которых выпускается версия для внутреннего или внешнего использования. Прохождение через четыре основные фазы называется циклом разработки, каждый цикл завершается генерацией версии системы. Если после этого работа над проектом не прекращается, то полученный продукт продолжает развиваться и снова минует те же фазы. Суть работы в рамках RUP - это создание и сопровождение моделей на базе UML.

- ***Microsoft Solution Framework (MSF)*** сходна с RUP, так же включает четыре фазы: анализ, проектирование, разработка, стабилизация, является итерационной, предполагает использование объектно-ориентированного моделирования. MSF в сравнении с RUP в большей степени ориентирована на разработку бизнес-приложений.

- ***Extreme Programming (XP).*** Экстремальное программирование (самая новая среди рассматриваемых методологий) сформировалось в 1996 году. В основе

методологии командная работа, эффективная коммуникация между заказчиком и исполнителем в течение всего проекта по разработке ИС, а разработка ведется с использованием последовательно дорабатываемых прототипов.

Основные процессы жизненного цикла ISO/IEC 12207

В соответствии с базовым международным стандартом ISO/IEC 12207 все процессы ЖЦ ПО делятся на три группы:

1. **Основные процессы:**
    - приобретение;
    - поставка;
    - разработка;
    - эксплуатация;
    - сопровождение.
2. **Вспомогательные процессы:**
    - документирование;
    - управление конфигурацией;
    - обеспечение качества;
    - разрешение проблем;
    - аудит;
    - аттестация;
    - совместная оценка;
    - верификация.
3. **Организационные процессы:**
    - создание инфраструктуры;
    - управление;
    - обучение;
    - усовершенствование.

В таблице 1 приведены ориентировочные описания основных процессов ЖЦ. Вспомогательные процессы предназначены для поддержки выполнения основных процессов, обеспечения качества проекта, организации верификации, проверки и тестирования ПО. Организационные процессы определяют действия и задачи, выполняемые как заказчиком, так и разработчиком проекта для управления своими процессами.

Для поддержки практического применения стандарта ISO/IEC 12207 разработан ряд технологических документов:

- Руководство для ISO/IEC 12207 (ISO/IEC TR 15271:1998 Information technology - Guide for ISO/IEC 12207)
- Руководство по применению ISO/IEC 12207 к управлению проектами (ISO/IEC TR 16326:1999 Software engineering - Guide for the application of ISO/IEC 12207 to project management).

Таблица 1. Содержание основных процессов ЖЦ ПО ИС (ISO/IEC 12207)

Процесс (исполнитель процесса)	**Действия**	**Вход**	**Результат**
Приобретение (заказчик)	• Инициирование • Подготовка заявочных предложений • Подготовка договора • Контроль деятельности поставщика • Приемка ИС	• Решение о начале работ по внедрению ИС • Результаты обследования деятельности заказчика • Результаты анализа рынка ИС/ тендера • План поставки/ разработки • Комплексный тест ИС	• Технико-экономическое обоснование внедрения ИС • Техническое задание на ИС • Договор на поставку/ разработку • Акты приемки этапов работы • Акт приемно-сдаточных испытаний
Поставка (разработчик ИС)	• Инициирование • Ответ на заявочные предложения • Подготовка договора • Планирование исполнения • Поставка ИС	• Техническое задание на ИС • Решение руководства об участии в разработке • Результаты тендера • Техническое задание на ИС • План управления проектом • Разработанная ИС и документация	• Решение об участии в разработке • Коммерческие предложения/ конкурсная заявка • Договор на поставку/ разработку • План управления проектом • Реализация/ корректировка • Акт приемно-сдаточных испытаний

Разработка (разработчик ИС)	• Подготовка • Анализ требований к ИС • Проектирование архитектуры ИС • Разработка требований к ПО • Проектирование архитектуры ПО • Детальное проектирование ПО • Кодирование и тестирование ПО • Интеграция ПО и квалификационное тестирование ПО • Интеграция ИС и квалификационное тестирование ИС	• Техническое задание на ИС • Техническое задание на ИС, модель ЖЦ • Подсистемы ИС • Спецификации требования к компонентам ПО • Архитектура ПО • Материалы детального проектирования ПО • План интеграции ПО, тесты • Архитектура ИС, ПО, документация на ИС, тесты	• Используемая модель ЖЦ, стандарты разработки • План работ • Состав подсистем, компоненты оборудования • Спецификации требования к компонентам ПО • Состав компонентов ПО, интерфейсы с БД, план интеграции ПО • Проект БД, спецификации интерфейсов между компонентами ПО, требования к тестам • Тексты модулей ПО, акты автономного тестирования • Оценка соответствия комплекса ПО требованиям ТЗ • Оценка соответствия ПО, БД, технического комплекса и комплекта документации требованиям ТЗ

Погонин – Интегрированные системы проектирования ИС (35-37).

ISO/IEC 12207 (ISO – International Organization of Standardization – Международная организация по стандартизации; IEC – International Electrotechnical Commission –

Международная комиссия по электротехнике) определяет структуру ЖЦ, содержащую процессы, действия и задачи, которые должны быть выполнены во время создания ПО.

Эксплуатация включает в себя работы по внедрению компонентов ПО в эксплуатацию, в том числе конфигурирование базы данных и рабочих мест пользователей, обеспечение эксплуатационной документацией, проведение обучения персонала и т.д., и непосредственно эксплуатацию, в том числе локализацию проблем и устранение причин их возникновения.

Управление проектом связано с вопросами планирования и организации работ, создания коллективов разработчиков и контроля за сроками и качеством выполняемых работ. Техническое и организационное обеспечение проекта включает выбор методов и инструментальных средств для реализации проекта, определение методов описания промежуточных состояний разработки, разработку методов и средств испытаний ПО.

Верификация – это процесс определения того, отвечает ли текущее состояние разработки, достигнутое на данном этапе, требованиям этого этапа. Проверка позволяет оценить соответствие параметров разработки исходным требованиям. Проверка частично совпадает с тестированием, которое связано с идентификацией различий между действительными и ожидаемыми результатами и оценкой соответствия характеристик ПО исходным требованиям.

Управление конфигурацией является одним из вспомогательных процессов, поддерживающих основные процессы жизненного цикла ПО, прежде всего процессы разработки и сопровождения ПО. При создании проектов сложных ИС, состоящих из многих компонентов, каждый из которых может иметь разновидности или версии, возникают проблемы учета их связей и функций, создания унифицированной структуры и обеспечения развития всей системы. Управление конфигурацией позволяет организовать внесение изменений в ПО на всех стадиях ЖЦ. Общие принципы и рекомендации конфигурационного учета, планирования и управления конфигурациями ПО отражены в проекте стандарта ***ISO 12207-2***.

Каждый процесс характеризуется определенными задачами и методами их решения, исходными данными, полученными на предыдущем этапе, и результатами. Результатами анализа, в частности, являются функциональные модели, информационные модели и соответствующие им диаграммы. ЖЦ ПО носит итерационный характер: результаты очередного этапа часто вызывают изменения в проектных решениях, выработанных на более ранних этапах.

Модели жизненного цикла ПО

Стандарт ISO/IEC 12207 не предлагает конкретную модель ЖЦ и методы разработки ПО (под моделью ЖЦ понимается структура, определяющая последовательность выполнения и взаимосвязи процессов, действий и задач, выполняемых на протяжении ЖЦ. Модель ЖЦ зависит от специфики ИС и условий, в которых последняя создается и функционирует). Его регламенты являются общими для любых моделей ЖЦ, методологий и технологий разработки. Стандарт ISO/IEC 12207 описывает структуру процессов ЖЦ ПО, но не конкретизирует в деталях, как реализовать или выполнить действия и задачи, включенные в эти процессы.

В изначально существовавших однородных ИС каждое приложение представляло собой единое целое. Для разработки такого типа приложений применялся каскадный способ. Его основной характеристикой является разбиение всей разработки на этапы, причем переход с одного этапа на следующий происходит только после того, как будет

полностью завершена работа на текущем (рис. 2.1). Каждый этап завершается выпуском полного комплекта документации, достаточной для того, чтобы разработка могла быть продолжена другой командой разработчиков.

Рис. 2.1 Каскадная схема разработки ПО

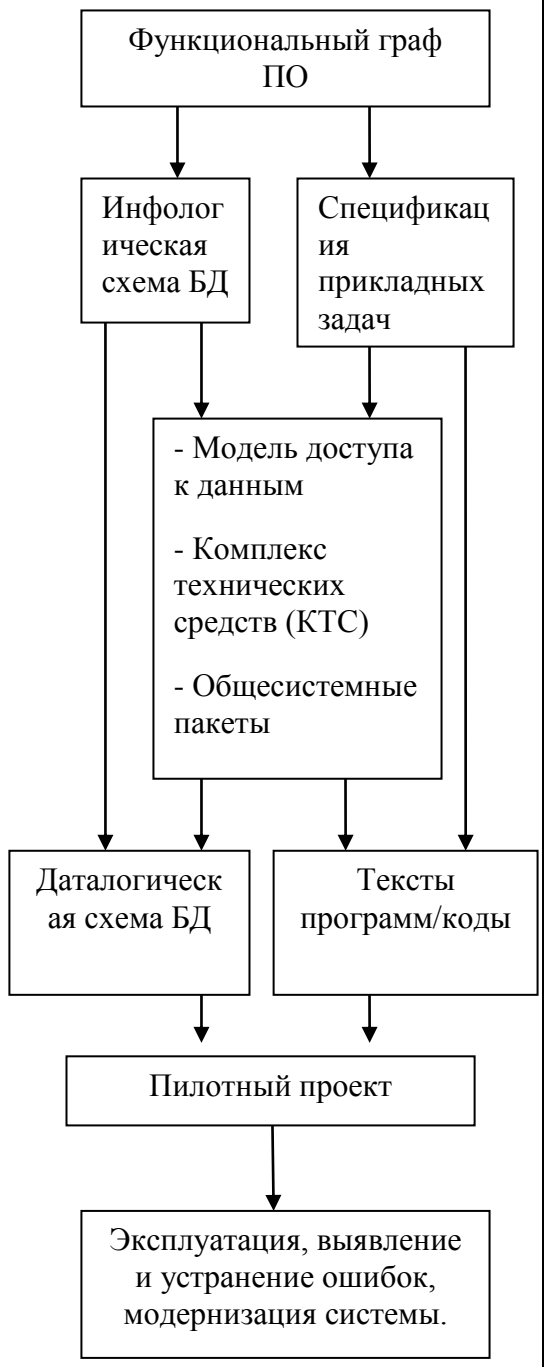
Положительные стороны применения каскадного подхода заключаются в следующем:

- на каждом этапе формируется законченный набор проектной документации, отвечающий критериям полноты и согласованности;
- выполняемые в логичной последовательности этапы работ позволяют планировать сроки завершения всех работ и соответствующие затраты.

Каскадный подход хорошо зарекомендовал себя при построении ИС, для которых в самом начале разработки можно достаточно точно и полно сформулировать все требования, с тем, чтобы предоставить разработчикам свободу реализовать их как можно лучше с технической точки зрения. В эту категорию попадают сложные расчетные системы, системы реального времени и другие подобные задачи. Однако в процессе использования этого подхода обнаружился ряд его недостатков, вызванных прежде всего тем, что реальный процесс создания ПО никогда полностью не укладывался в такую жесткую схему. В процессе создания ПО постоянно возникала потребность в возврате к предыдущим этапам и уточнении или пересмотре ранее принятых решений. В результате реальный процесс создания ПО принимал вид, представленный на рис. 2.2.

Рис. 2.2 Реальный процесс разработки ПО по каскадной схеме

Рассмотрим более подробно каскадную модель:

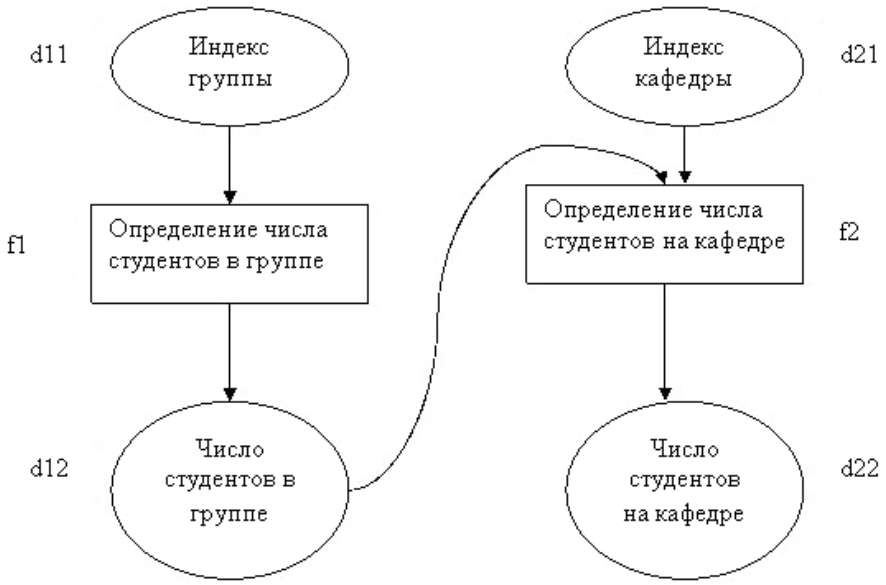
Этап	Результат
1. Выявление информационных потребностей конечных пользователей (предпроектное обследование, разработка ТЗ, частных ТЗ) 2. Концептуальный проект (эскизное проектирование) 3. Архитектура системы (технический проект) 4. Логическое проектирование (техническое проектирование) 5. Комплексная отладка (разработка рабочей документации) 6. Сопровождение.	

Особенность модели: каждый следующий этап проектирования начинается после полного завершения работ по предыдущему этапу.

1. Выявление информационных потребностей конечных пользователей

Функциональный граф ПО - граф, узлы которого обозначают данные и процессы будущей системы. Дуги используются для обозначения входных/выходных данных для процесса.

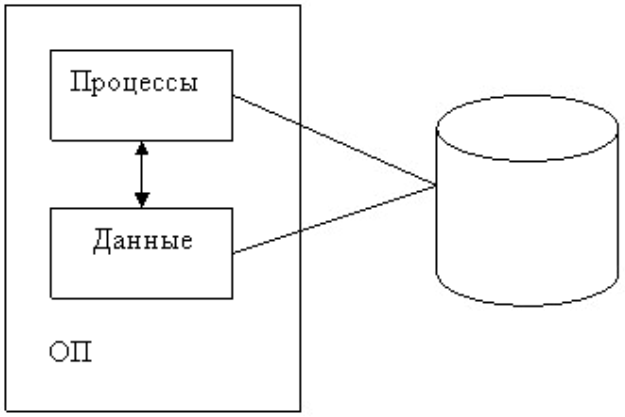
Пример:



d22=f2(d12,d21)=f2(f1(d11),d21)

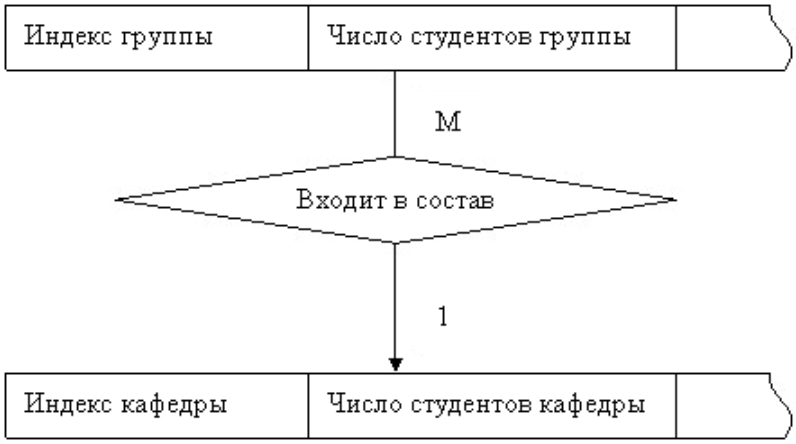
В функциональном графе данные и процессы объединены и в принципе его достаточно для реализации будущей системы (см. формулу к рисунку), но современные компьютеры есть машины Фон-Неймана, где предполагается разделение процессов и данных.

2. Концептуальный проект



Для реализации концептуальной модели проектировщик вынужден выделять из функционального графа данные и строить для них схему БД, а также выделять процессы и разрабатывать для них спецификацию и кодировать. Это является источником большинства ошибок проектирования. Данные функционального графа структурируются в виде инфологической схемы БД.

Пример: (Инфологическая модель БД в нотации Чена)



Спецификация процессов - входные и выходные данные процессов, а также алгоритмическая связь между ними. Для описания спецификации существуют различные методы: структурированный естественный язык (часто используется), язык проектирования спецификации Flow-Form (визуальные языки).

Концептуальный проект не зависит от архитектуры!

3. Выбор архитектуры

- выбор модели доступа к данным (файл-сервер, сервер-БД, сервер-приложение, доступ к данным по Internet/Intarnet)

- выбор комплекса технических средств (выбор «железа»)

- выбор общесистемных пакетов

-выбор способа тиражирования данных

4. Логическое проектирование

Выполняется отражение концептуального проекта в СУБД-ориентированную среду с помощью выбранных оболочек программирования. Сущности преобразуются в таблицы, а на основе спецификации задач разрабатываются тексты программ

Логический проект зависит от архитектуры (можно считать временные характеристики)

5. Отладка

Результаты проектирования БД и приложений объединяются. В итоге разрабатывается пилотный проект системы

6. Сопровождение

Выявление ошибок и их устранение, модернизация.

Достоинства каскадной модели:

- проста, естественна, имеет некоторую привязку к ГОСТу

Недостатки:

- достаточно продолжительный цикл разработки по времени (система морально устаревает)

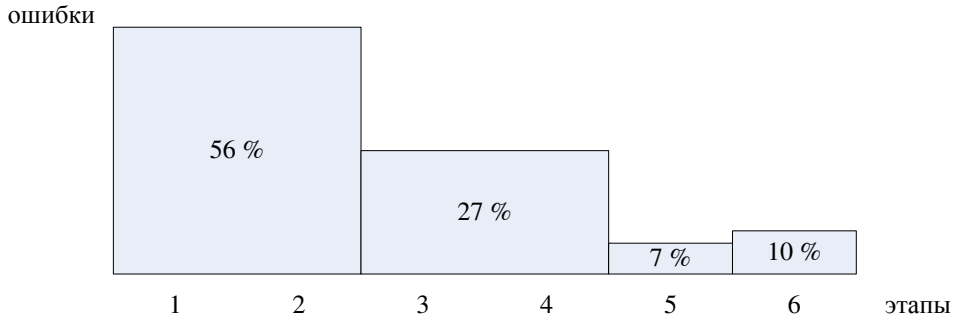
- доработка системы связана с большим объемом перепрограммирования (из-за слабого использования CASE-средств)

Основным недостатком каскадного подхода является существенное запаздывание с получением результатов. Согласование результатов с пользователями производится только в точках, планируемых после завершения каждого этапа работ, требования к ИС «заморожены» в виде технического задания на все время ее создания.

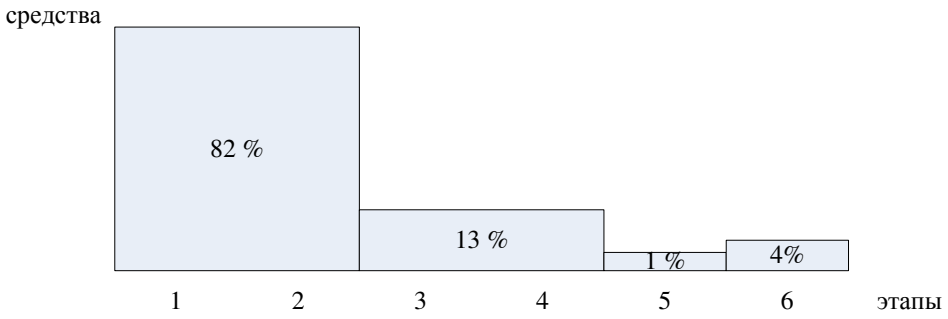
Таким образом, пользователи могут внести свои замечания только после того, как работа над системой будет полностью завершена. В случае неточного изложения требований или их изменения в течение длительного периода создания ПО пользователи получают систему, не удовлетворяющую их потребностям. Модели (как функциональные, так и информационные) автоматизируемого объекта могут устареть одновременно с их утверждением.

Результаты исследований Д.Мартина (сер. 80х)

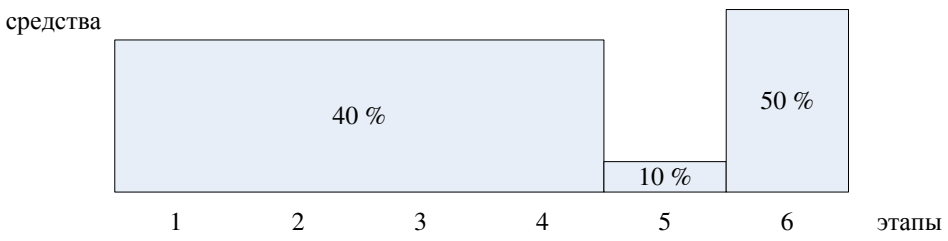
1-я диаграмма: распределение ошибок и просчетов по этапам проектирования, выявленных при сопровождении системы



2-я диаграмма: распределение затрат на исправление ошибок и просчетов, выявленных при сопровождении



3-я диаграмма: распределение трудозатрат по этапам проектирования



Почти половина трудозатрат приходится на устранение ошибок, допущенных на первых 2-х этапах.

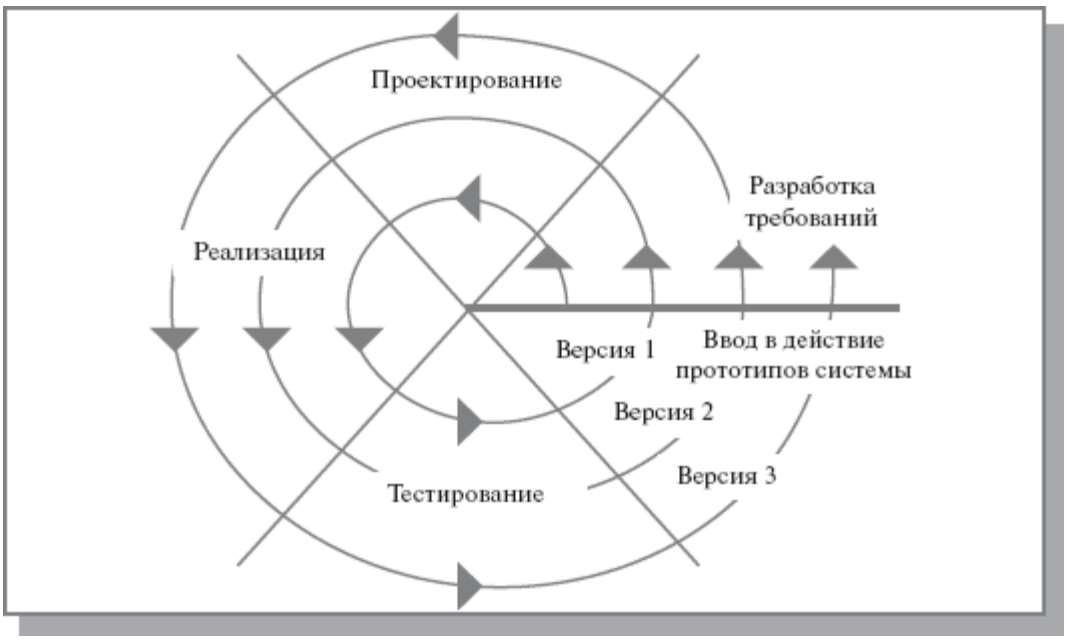
На основании исследований Мартин сформулировал законы:

1. **закон неопределенности в информатике**: процесс автоматизации задачи меняет представление пользователя об этой задаче, т.е. пользователь решает задачу с использованием средств автоматизации иначе, чем без них (пользователя надо использовать постоянно в процессе проектирования, а не только в начале)
2. чем больше времени прошло с момента совершения ошибки до момента ее обнаружения, тем больше средств необходимо для ее устранения (смотри диаграмму 2)
3. программисты и проектировщики не учатся на чужих ошибках, а только на своих.

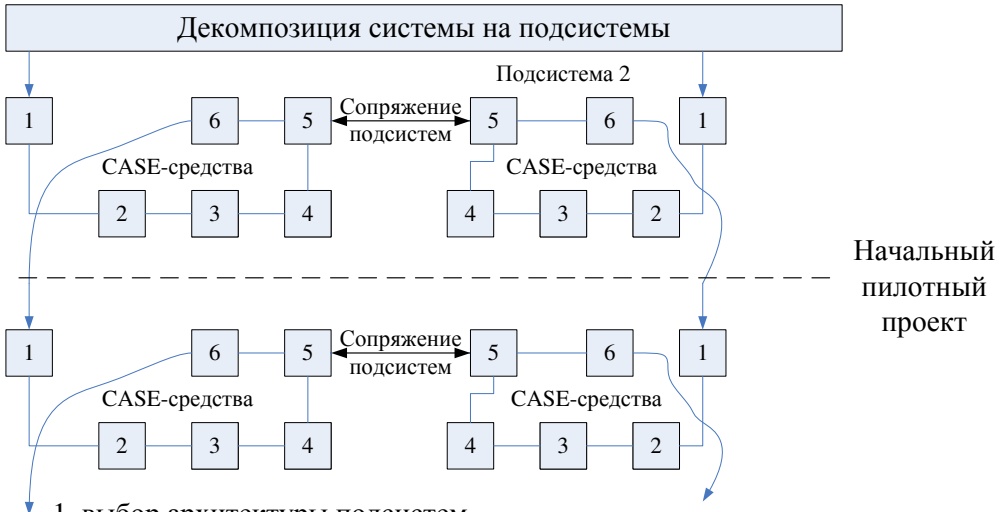
Спиральная модель

Для преодоления перечисленных проблем была предложена спиральная модель ЖЦ делающая упор на начальные этапы ЖЦ: анализ и проектирование. На этих этапах реализуемость технических решений проверяется путем создания прототипов. Каждый виток спирали соответствует созданию фрагмента или версии ПО, на нем уточняются цели и характеристики проекта, определяется его качество и планируются работы следующего витка спирали. Таким образом углубляются и последовательно конкретизируются детали проекта и в результате выбирается обоснованный вариант, который доводится до реализации.

Разработка итерациями отражает объективно существующий спиральный цикл создания системы. Неполное завершение работ на каждом этапе позволяет переходить на следующий этап, не дожидаясь полного завершения работы на текущем. При итеративном способе разработки недостающую работу можно будет выполнить на следующей итерации. Главная же задача – как можно быстрее показать пользователям системы работоспособный продукт, тем самым активизируя процесс уточнения и дополнения требований.



Основная проблема спирального цикла – определение момента перехода на следующий этап. Для ее решения необходимо ввести временные ограничения на каждый из этапов жизненного цикла. Переход осуществляется в соответствии с планом, даже если не вся запланированная работа закончена. План составляется на основе статистических данных, полученных в предыдущих проектах, и личного опыта разработчиков.



1. выбор архитектуры подсистем
2. выявление информационных потребностей конечных пользователей системы
3. концептуальное проектирование
4. логическое проектирование
5. отладка подсистем
6. сопровождение подсистемы

Каждый этап реализуется CASE-средствами. Содержание этапов совпадает с аналогичными в каскадной модели, но в отличие от нее, этапы реализуются с помощью CASE-средств (Computer Aided Software System Design) – **(автоматизированная технология создания программных информационных систем)** – использование этих средств позволяет существенно снизить время реализации витка спирали проектирования

подсистем (в этом состоит основное преимущество спиральной модели), но это профессиональные средства, непредназначенные для конечных пользователей.

С помощью CASE-средств можно быстро сгенерировать проект хотя бы на уровне экранных форм и показать его конечному пользователю, который высказывает свои предложения и замечания, и на следующем витке реализуются они. Когда спецификация на уровне экранных форм будет будет согласована, то проектировщик может начинать детальную реализацию. Описанная схема разработки называют визуальным проектированием.

Витков может быть много, разделение на этапы условно. Работы могут выполняться параллельно или в комплексной итерации на любом витке спирали.

Проектирование как конструирование

Идея конструирования информационных систем (ИС) из компонентов развивается на протяжении многих лет. По существу, речь идет о крупноблочном конструировании систем (Д. Видерхольд). Согласно этой идее, целесообразнее осуществлять капиталовложения в создание компонентов, которые можно было бы многократно использовать, чем всякий раз осуществлять разработку ИС сверху – вниз: от спецификаций требований к работающей системе.

Во-первых, следует отметить то обстоятельство, что компонентно-ориентированный подход к проектированию и реализации программных систем и комплексов является в некотором смысле развитием объектно-ориентированного и практически более пригоден для разработки крупных и распределенных систем (например, корпоративных приложений).

Прежде всего, сформулируем основополагающее для рассматриваемого подхода определение компонента. Под компонентом будем далее иметь в виду независимый модуль программного кода, предназначенный для повторного использования и развертывания. Как видно из определения, применение компонентного программирования призвано обеспечить более простую, быструю и прямолинейную процедуру первоначальной инсталляции прикладного программного обеспечения, а также увеличить процент повторного использования кода, т.е. усилить основные преимущества ООП.

Говоря о свойствах компонентов, следует прежде всего отметить, что это существенно более крупные единицы, чем объекты (в том смысле, что объект представляет собой конструкцию уровня языка программирования). Другими отличиями компонентов от традиционных объектов являются возможность содержать множественные классы и (в большинстве случаев) независимость от языка программирования.

Естественным требованием к современным информационным системам является способность наращивания их возможностей за счет использования дополнительно разработанных (а еще лучше - уже существующих) программных компонентов. Для этого требуется обеспечение свойства, называемого интероперабельностью.

Интероперабельность программного обеспечения (функциональность программного обеспечения) - способность программного продукта выполнять набор функций, определенных в его внешнем описании и удовлетворяющих заданным или подразумеваемым потребностям пользователей.

Под этим понимается соблюдение определенных правил или привлечение дополнительных программных средств, обеспечивающих возможность взаимодействия независимо от разработанных программных модулей, подсистем или даже функционально завершенных программных систем.

Новейшие технологии обеспечивают техническую возможность интероперабельного использования как программных (OMG CORBA), так и информационных компонент (WWW).

Аспекты описания систем

Наряду с декомпозицией описаний на иерархические уровни применяют разделение представлений о проектируемых объектах на аспекты.

Расчленение (декомпозиция) описаний по характеру отображаемых свойств объекта приводит к появлению ряда аспектов описания.

Аспект описания (страта) — описание системы или ее части с некоторой оговоренной точки зрения, определяемой функциональными, физическими или иного типа отношениями между свойствами и элементами.

Различают функциональный, информационный, структурный и поведенческий (процессный) аспекты.

Функциональное описание относят к функциям системы и чаще всего представляют его функциональными схемами.

Информационное описание включает в себя основные понятия предметной области (сущности), словесное пояснение или числовые значения характеристик (атрибутов) используемых объектов, а также описание связей между этими понятиями и характеристиками.

Информационные модели можно представлять графически (графы, диаграммы сущность - отношение), в виде таблиц или списков.

Структурное описание относится к морфологии системы, характеризует составные части системы и их межсоединения и может быть представлено структурными схемами, а также различного рода конструкторской документацией.

Поведенческое описание характеризует процессы функционирования (алгоритмы) системы и (или) технологические процессы создания системы. Иногда аспекты описаний связывают с подсистемами, функционирование которых основано на различных физических процессах.

Отметим, что в общем случае выделение страт может быть неоднозначным. Так, помимо указанного подхода очевидна целесообразность выделения таких аспектов, как функциональное (разработка принципов действия, структурных, функциональных, принципиальных схем), конструкторское (определение форм и пространственного расположения компонентов изделий), алгоритмическое (разработка алгоритмов и программного обеспечения) и технологическое (разработка технологических процессов) проектирование систем.

Типовые проектные решения – шаблоны проектирования

Идея, на которой основывается применение шаблонов проектирования, заключается в том, чтобы выработать стандартизованный подход к представлению общих решений, пригодных для часто встречающихся ситуаций при разработке ПО. Такой подход имеет ряд преимуществ.

- Со временем можно получить каталог шаблонов. Это позволяет новичкам в разработке ПО более эффективно использовать многолетний опыт их предшественников.

- Все решения, принимаемые при проектировании ПО, снабжены формализованным описанием, позволяющим оценить как достоинства, так и недостатки того или иного решения. Стандартизованные шаблоны облегчают как для новичков, так и для экспертов в области разработки ПО понимание того, как влияют на архитектуру создаваемой программы те или иные решения.

- Шаблоны проектирования позволяют всем, кто их использует, "говорить на одном языке". Это, в свою очередь, облегчает взаимопонимание между разработчиками при выборе того или иного решения. Вместо того чтобы детально описывать какое-либо архитектурное решение, мы говорим, какой шаблон мы намерены использовать.

Мы можем соотносить шаблоны друг с другом, что позволяет разработчику увидеть, какие шаблоны могут использоваться в одном проекте.

Шаблоны проектирования предоставляют в наше распоряжение эффективный способ делиться опытом со всеми участниками сообщества объектно-ориентированного программирования. Независимо от того, каким языком программирования мы владеем (C++, Smalltalk или Java), и того, в какой области мы приобрели опыт проектирования ПО (Web-проекты, интеграция устаревших систем или заказной проект), мы можем накапливать свой собственный опыт и делиться им с другими разработчиками. Если же говорить о долгосрочной перспективе, то данный подход позволяет улучшить состояние дел во всей индустрии разработки ПО.

История применения шаблонов

Идейным отцом применения шаблонов проектирования при разработке ПО считают профессора архитектуры Университета Калифорнии в Беркли Кристофера Александера (Christofer Alexander). В конце 70-х годов прошлого века он опубликовал несколько книг, в которых изложил основные принципы применения шаблонов в архитектуре, а также поместил каталог архитектурных шаблонов.

Именно посвященные шаблонам работы Александера и привлекли внимание к проблеме программистов, интересующихся объектно-ориентированным программированием (ООП). В их среде появились пионеры применения шаблонов в разработке ПО, которые на протяжении последующих десяти лет сформулировали основные принципы этого метода. Среди первопроходцев были Кент Бэк (Kent Beck) и Уард Каннингхэм (Ward Cunningham). В 1987 году на конференции OOPSIA (Object Oriented Programming, Systems, and Applications), посвященной ООП, прозвучал их доклад о применении шаблонов проектирования в языке Smalltalk. Еще одним адептом нового подхода стал Джеймс Коплайен (James Coplien), который в начале 90-х гг написал книгу о применении идиом (т.е. шаблонов) при разработке ПО на языке C++.

Ежегодные конференции OOPSLA сослужили хорошую службу для роста сторонников технологии шаблонов, так как на ее мероприятиях энтузиасты могли свободно делиться своими идеями со множеством благодарных слушателей. Кроме того, важную роль в

становлении данного направления технологии разработки ПО сыграли конференции некоммерческой организации Hillside Group, создателями которой были Кент Бэк и Гради Буч (Grady Booch).

Однако по-настоящему ощутимый вклад в дело популяризации технологии шаблонов проектирования внесла изданная в 1995 году книга *Design Patterns: Elements of Reusable Object-Oriented Software.* Ее авторы Эрик Гамма (Erich Gamma), Ричард Хелм (Richard Helm), Ральф Джонсон (Ralph Johnson) и Джон Влиссидес (John Vlissides) приложили так много усилий для распространения своих идей, что заслужили шутливое прозвище "Банда четырех" (GoF — gang of four). В книге представлено введение в довольно сложный язык шаблонов с иллюстрациями реализации обсуждаемых шаблонов на языке C++.

В это же время началось бурное развитие технологии Java, поэтому неудивительно, что Java-разработчики с самого начала стали применять шаблоны проектирования в своих проектах. Рост популярности технологии шаблонов проектирования среди Java-разработчиков проявился как в виде специальных презентаций на конференциях, таких как JavaOne, так и в появлении отдельной рубрики по шаблонам проектирования практически во всех специализированных журналах по программированию на Java.

Основные концепции технологии шаблонов

В центре технологии шаблонов лежит идея стандартизации информации о типичной проблеме и о методах ее решения. Один из самых полезных результатов, полученных в работе Александера, состоит в выделении некоторого способа представления шаблонов, который впоследствии был назван *формой* (form), или *форматом* (format). В форме Александера применялось пять направлений, по которым формализовалось обсуждение шаблонов и их применение в конкретных ситуациях.

Самое главное преимущество такого подхода состоит в том, что даже само название шаблона представляет собой ответ на вопрос: "Что можно сделать с помощью этого шаблона?" Кроме того, в форме содержится: описание проблемной области; пояснение того, как данный шаблон решает означенную проблему; в чем заключаются преимущества, недостатки и, возможно, компромиссы при использовании данного шаблона.

Естественно, когда шаблоны были восприняты миром ООП, появились вариации формы Александера, призванные учитывать интересы разработчиков ПО. Большинство из существующих сегодня форм были построены на одной или двух формах, получивших название "Канонических", или форм "Четверки". В данной книге используется одна из вариаций форм "Четверки", согласно которой при представлении шаблонов освещаются следующие вопросы.

- *Название (Name).* Слово или выражение, описывающее основное назначение шаблона.
- *Также известен как (Also Known As).* Альтернативные названия (если, конечно, таковые имеются).
- *Свойства шаблона (Pattern Properties).* Классификация шаблона. Мы будем определять шаблон, используя термины из двух следующих групп.

Тип:

- *Производящие шаблоны (Creational patterns),* предназначенные для создания объектов.

- *Поведенческие шаблоны (Behavioral patterns),* обеспечивающие координацию функционального взаимодействия между объектами.
- *Структурные шаблоны (Structural patterns)*, используемые для управления статическими, структурными связями между объектами.
- *Системные шаблоны (System patterns)*, предназначенные для управления взаимодействием на системном уровне.

Уровень:

- *Отдельный класс (single class).* Шаблон применяется к отдельному, независимому классу.
- *Компонент (component).* Шаблон используется для создания группы классов.
- *Архитектурный (architectural).* Шаблон используется для координации работы систем и подсистем.

Повторное использование и абстракция программного кода

Технология шаблонов проектирования стала важным эволюционным этапом в развитии таких концепций, как *абстракция (abstraction)* и *повторное использование (reuse)* программного кода. Обе эти концепции занимают центральное место в самой идее программирования (некоторые даже полагают, что они являются важнейшими концепциями).

Абстракция — это метод, с помощью кот`орого разработчики могут решать сложные проблемы, последовательно разбивая их на более простые. Затем можно использовать имеющиеся готовые решения полученных типовых простых проблем в качестве строительных блоков, из которых разработчики получают решения, пригодные для реализации повседневных сложных проектов.

В ходе развития объектно-ориентированных языков программирования был совершен огромный скачок вперед в области абстракции и повторного использования программного кода. С помощью этой технологии была создана целая плеяда высокопроизводительных методов его создания.

Например, чего стоит только одна концепция класса как "эталона" объектов, с ее объединением двух ранее возникших механизмов: функциональной абстракции и абстракции данных. Объединяя структуру сущности (данные) с функциональностью, которая применяется к сущности (поведение), мы получаем эффективный метод повторного использования программного элемента.

Помимо основополагающего понятия класса, объектно-ориентированные языки принесли с собой множество других методов использования существующего кода.

В качестве примера можно привести такие концепции, как концепции подклассов и интерфейсов, открывших совершенно новые возможности в повторном использовании программного кода при разработке ПО.

Метод	*Повторное использование*	*Абстракция*	*Универсальность подхода*
Копирование и вставка	Очень плохо	Отсутствует	Очень плохо
Структуры данных	Хорошо	Тип данных	Средне — хорошо
Функциональность	Хорошо	Метод	Средне — хорошо
Типовые блоки кода	Хорошо	Типизируемая операция	Хорошо
Алгоритмы	Хорошо	Формула	Хорошо
Классы	Хорошо	Данные + метод	Хорошо
Библиотеки	Хорошо	Функции	Хорошо — очень хорошо
API	Хорошо	Классы утилит	Хорошо — очень хорошо
Компоненты	Хорошо	Группы классов	Хорошо — очень хорошо
Шаблоны проектирования	Отлично	Решения проблем	Очень хорошо

В столбце "Абстракция" таблицы указаны сущности, для которых выполняется абстракция, а в столбце "Универсальность подхода" показано то, насколько легко применить соответствующий метод, не прибегая к изменениям или переработке кода. Обратите внимание на то, что показатель степени повторного использования очень сильно зависит от эффективности применения того или иного метода на практике. Это понятно, так как любая, даже самая лучшая возможность может быть использована как правильно, так и неправильно.

Одной из самых выдающихся возможностей, предоставляемых шаблонами проектирования, является то, что они позволяют разработчикам более эффективно применять другие методы повторного использования программного кода. Шаблон в определенных ситуациях может, например, использоваться как руководство для эффективного управления наследованием или же для эффективного назначения связей между классами, обеспечивая тем самым решение той или иной проблемы.

Производящие шаблоны

Производящие шаблоны (creational patterns) предназначены для обеспечения выполнения одной из самых распространенных задач в ООП — создания объектов в системе.

В ходе работы большинства объектно-ориентированных систем, независимо от уровня их сложности, создается множество экземпляров объектов. Производящие шаблоны облегчают процесс создания объектов, предоставляя разработчику следующие возможности:

- Единый способ получения экземпляров объектов. В системе обеспечивается механизм создания объектов без необходимости идентификации определенных типов классов в программном коде.

- Простота. Некоторые из шаблонов упрощают процесс создания объектов до такой степени, что полностью избавляют разработчика от необходимости написания большого и сложного программного кода для получения экземпляра объекта.

- Учет ограничений при создании. Некоторые шаблоны позволяют при создании объектов налагать ограничения на их тип или количество в соответствии с установленными требованиями системы.

Основные производящие шаблоны проектирования.

- Abstract Factory. Обеспечивает создание семейств взаимосвязанных или зависящих друг от друга объектов без указания их конкретных классов.

- Builder. Упрощает создание сложных объектов путем определения класса, предназначенного для построения экземпляров другого класса. Шаблон Builder генерирует только одну сущность. Хотя эта сущность в свою очередь может содержать более одного класса, но один из полученных классов всегда является главным.

- Factory Method. Определяет стандартный метод создания объекта, не связанный с вызовом конструктора, оставляя решение о том, какой именно объект создавать, за подклассами.

- Prototype. Облегчает динамическое создание путем определения классов, объекты которых могут создавать собственные дубликаты.

- Singleton. Обеспечивает наличие в системе только одного экземпляра заданного класса, позволяя другим классам получать доступ к этому экземпляру.

Два из перечисленных выше шаблона, а именно Abstract Factory и Factory Method, базируются исключительно на концепции определения создания настраиваемых объектов. Подразумевается, что разработчик, применяющий эти шаблоны, при реализации системы обеспечит механизм расширения создаваемых классов или интерфейсов. В силу этой особенности данные шаблоны часто объединяются с другими производящими шаблонами.

Поведенческие шаблоны

Поведенческие шаблоны (behavioral patterns) применяются для передачи управления в системе. Существуют методы организации управления, применение которых позволяет добиться значительного повышения как эффективности системы, так и удобства ее эксплуатации. Поведенческие шаблоны представляют собой проверенные на практике методы и обеспечивают понятные и простые в применении эвристические способы организации управления.

В данной главе рассматриваются следующие поведенческие шаблоны.

- Chain of Responsibility. Предназначен для организации в системе уровней ответственности. Использование этого шаблона позволяет установить, должно ли сообщение обрабатываться на том уровне, где оно было получено, или же оно должно передаваться для обработки другому объекту.

- Command. Обеспечивает обработку команды в виде объекта, что позволяет сохранять ее, передавать в качестве параметра методам, а также возвращать ее в виде результата, как и любой другой объект.

- Interpreter. Определяет интерпретатор некоторого языка.

- Iterator. Предоставляет единый метод последовательного доступа к элементам коллекции, не зависящий от самой коллекции и никак с ней не связанный.

- Mediator. Предназначен для упрощения взаимодействия объектов системы путем создания специального объекта, который управляет распределением сообщений между остальными объектами.

- Memento. Сохраняет "моментальный список" состояния объекта, позволяющий такому объекту вернуться к исходному состоянию, не раскрывая своего содержимого внешнему миру.

- Observer. Предоставляет компоненту возможность гибкой рассылки сообщений интересующим его получателям.

- State. Обеспечивает изменение поведения объекта во время выполнения программы.

- Strategy. Предназначен для определения группы классов, которые представляют собой набор возможных вариантов поведения. Это дает возможность гибко подключать те или иные наборы вариантов поведения во время работы приложения, меняя его функциональность "на ходу".

- Visitor. Обеспечивает простой и удобный в эксплуатации способ выполнения тех или иных операций для определенного семейства классов. Это достигается путем централизации с помощью данного шаблона возможных вариантов поведения, что позволяет модифицировать или расширять их, не затрагивая классы, на которые распространяются эти варианты поведения.

- Template Method. Предоставляет метод, который позволяет подклассам перекрывать части метода, не прибегая к их переписыванию.

Структурные шаблоны

Структурные шаблоны (structural patterns) с одинаковой эффективностью применяются как для разделения, так и для объединения элементов приложения. Способы воздействия структурных шаблонов на приложение могут быть самые разные.

Например, шаблон Adapter может обеспечить возможность двум несовместимым системам обмениваться информацией, тогда как шаблон Facade позволяет отобразить упрощенный пользовательский интерфейс, не удаляя ненужных конкретному пользователю элементов управления.

Структурные шаблоны, рассмотренные в данной главе, имеют следующее назначение.

- Adapter. Обеспечение взаимодействия двух классов путем преобразования интерфейса одного из них таким образом, чтобы им мог пользоваться другой класс.

- Bridge. Разделение сложного компонента на две независимые, но взаимосвязанные иерархические структуры: функциональную абстракцию и внутреннюю реализацию. Это облегчает изменение любого аспекта компонента.

- Composite. Предоставление гибкого механизма для создания иерархических древовидных структур произвольной сложности, элементы которых могут свободно взаимодействовать с единым интерфейсом.

- Decorator. Предоставление механизма для добавления или удаления функциональности компонентов без изменения их внешнего представления или функций.
- Facade. Создание упрощенного интерфейса для группы подсистем или сложной подсистемы.
- Flyweight. Уменьшение количества объектов системы с многочисленными низкоуровневыми особенностями путем совместного использования подобных объектов.
- Half-Object Plus Protocol (HOPP). Предоставление единой сущности, которая размещается в двух или более областях адресного пространства.
- Proxy. Представление другого объекта, обусловленное необходимостью обеспечения доступа или повышения скорости либо соображениями безопасности

Источники

1. Грекул – Проектирование ИС Лекция 2.
2. Стелтинг, Маассен – Шаблоны проектирования в Java (16-19)
   Производящие шаблоны (стр.25-26)
   Прототип (стр. 48-51)
   Поведенческие шаблоны (стр. 61-62)
   Итератор (стр. 88-90)
   Наблюдатель (стр. 111-113)
   Структурные шаблоны (стр. 155)
   Фасад (стр. 189-191)

Абстрактная фабрика

Порождающий шаблон проектирования, позволяющий изменять поведение системы, варьируя создаваемые объекты, при этом сохраняя интерфейсы.

Предоставляет интерфейс для создания семейств взаимосвязанных или взаимозависимых объектов, не специфицируя их конкретных классов.

Плюсы:

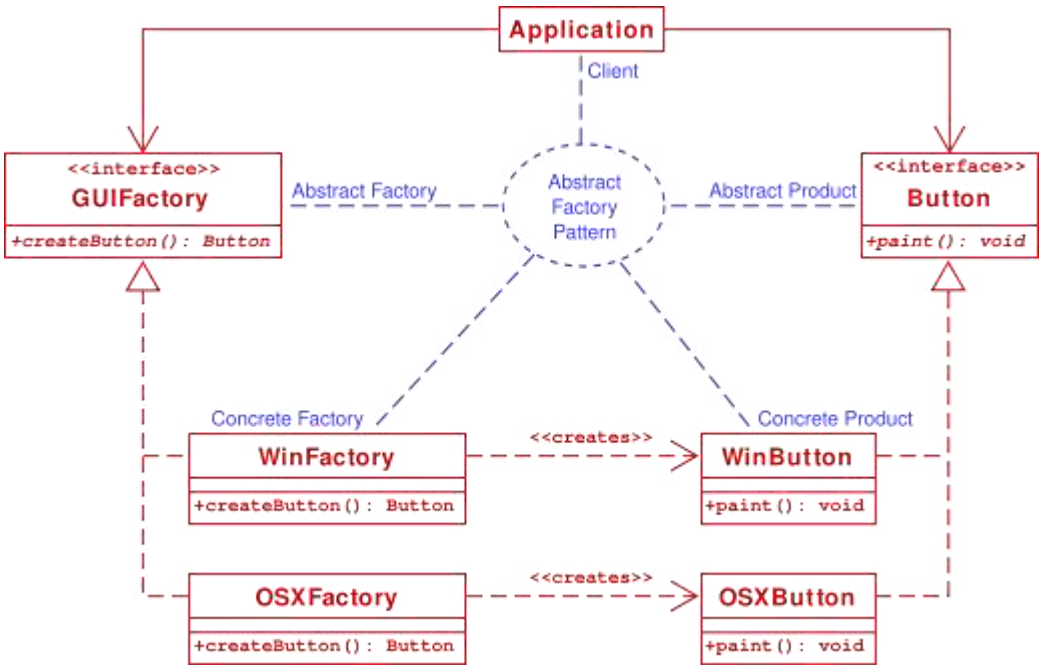
изолирует конкретные классы

упрощает замену семейств продуктов

гарантирует сочетаемость продуктов

Минусы:

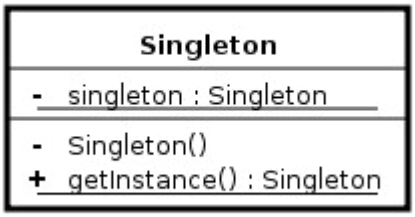
сложно добавить поддержку нового вида продуктов.



Синглтон

Объект-одиночка

Гарантирует, что у класса есть только один экземпляр, и предоставляет к нему глобальную точку доступа.



Плюсы:

контролируемый доступ к единственному экземпляру

уменьшение числа имён

допускает уточнение операций и представления

допускает переменное число экземпляров

бóльшая гибкость, чем у операций класса

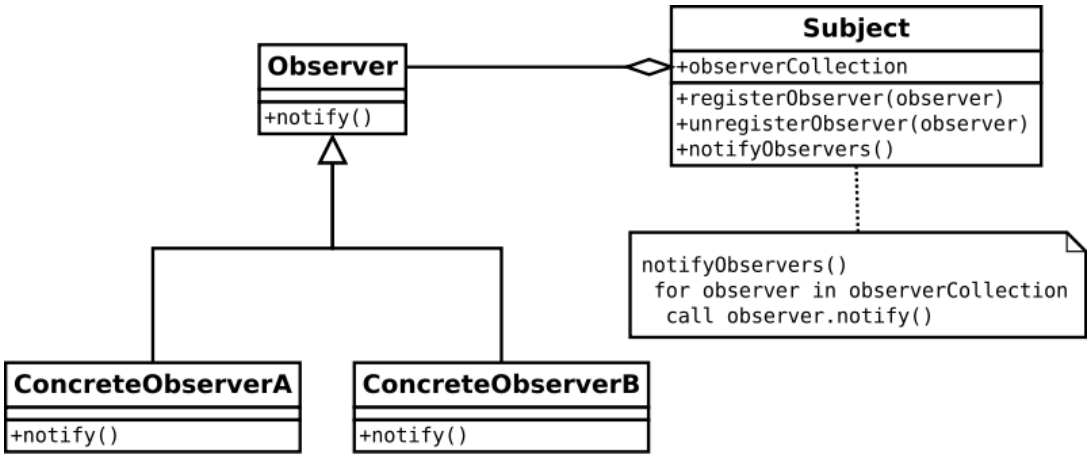
Минусы:

Глобальные объекты могут быть вредны для объектного программирования, в некоторых случаях приводя к созданию немасштабируемого проекта.

Усложняет написание модульных тестов

Наблюдатель

Поведенческий шаблон. Определяет зависимость типа «один ко многим» между объектами таким образом, что при изменении состояния одного объекта все зависящие от него оповещаются об этом событии.



Пример кода в отдельном файле observer.py

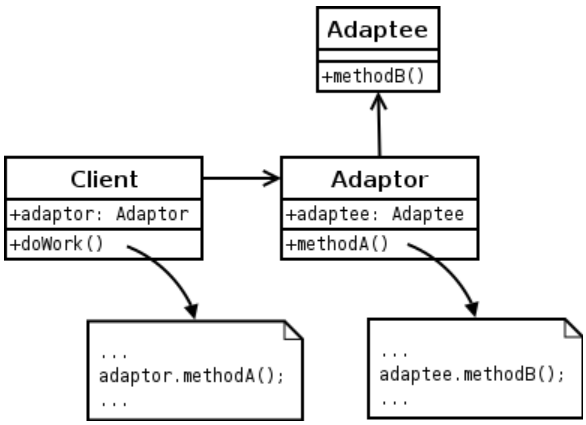
Адаптер

Структурный шаблон, предназначенный для организации использования функций объекта, недоступного для модификации, через специально созданный интерфейс.

Задача: Система поддерживает требуемые данные и поведение, но имеет неподходящий

интерфейс.

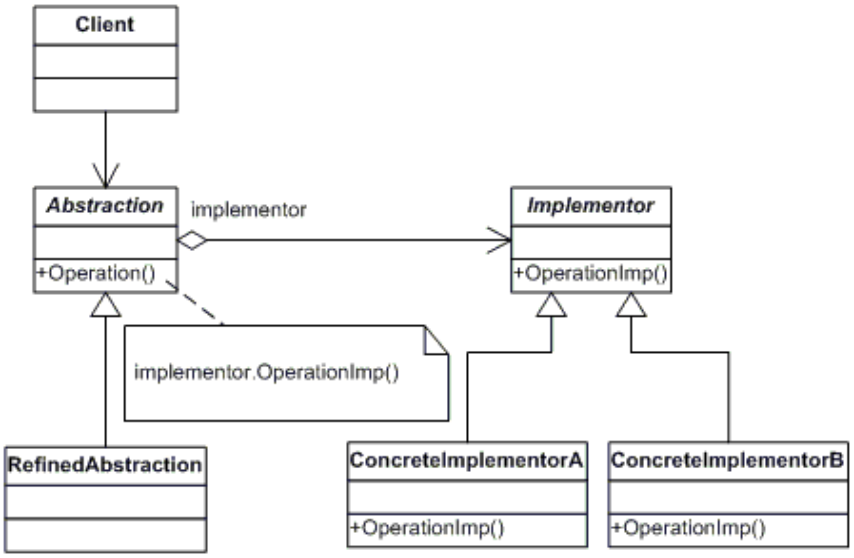
Решение:Адаптер предусматривает создание класса-оболочки с требуемым интерфейсом




```
class Adaptee:
    def specific_request(self):
        return 'Adaptee'
class Adapter:
    def __init__(self, adaptee):
        self.adaptee = adaptee
    def request(self):
        return self.adaptee.specific_request()
client = Adapter(Adaptee())
    print client.request()
```

Мост

Структурный шаблон, предназначенный для того, чтобы «разделять абстракцию и реализацию так, чтобы они могли изменяться независимо».



Пример в отдельном файле bridge.py.

Приложение 2

Сложность алгоритмов

Понятие сложности алгоритма входит в теорию алгортмов, которая изучает общие свойства и закономерности алгоритмов, а также разнообразные модели их представления.

Задачи теории алгоритмов:

- Асимптотический анализ сложности алгоритмов
- Доказательство асимптотической неразрешимости алгоритмов
- Классификация алгоритмов в соответствии с классами сложности

Понятие сложности также является одним из основных объектов изучения в теории сложности вычислений, изучающей стоимость работы, требуемой для решения вычислительной проблемы.

Стоимость обычно измеряется такими абстрактными понятиями, как *время* и *пространство*, называемыми вычислительными ресурсами.

Время – количество элементарных шагов, необходимых для решения задачи. Время является основным параметром, характеризующим быстродействие алгоритма. Оно также называется *вычислительной сложностью*. Обычно учитывается худшее, среднее и лучшее время алгоритма.

Количество элементарных операций, затраченных алгоритмом для решения конкретного экземпляра задачи, зависит не только от размера входных данных, но и от самих данных. Например, количество операций алгоритма сортировки вставками значительно меньше в случае, если входные данные уже отсортированы. Чтобы избежать подобных трудностей, рассматривают понятие временной сложности алгоритма в худшем случае.

Временная сложность алгоритма (в худшем случае) — это функция размера входных и выходных данных, равная максимальному количеству элементарных операций, проделываемых алгоритмом для решения экземпляра задачи указанного размера (верхний предел). Располагая этим значением, мы точно знаем, что для выполнения алгоритма не потребуется большее количество времени. В задачах, где размер выхода не превосходит или пропорционален размеру входа, можно рассматривать временную сложность как функцию размера только входных данных.

Пример: Поиск информации по БД-> наихудший случай будет тогда, когда информация в БД отсутствует.

Аналогично понятию временной сложности в худшем случае определяется понятие временная сложность алгоритма в наилучшем случае. Также рассматривают понятие среднее время работы алгоритма, то есть математическое ожидание времени работы алгоритма. Иногда говорят просто: «Временная сложность алгоритма» или «Время работы алгоритма», имея в виду временную сложность алгоритма в худшем, наилучшем или среднем случае (в зависимости от контекста).

Пространство – объем памяти или места на носителе данных. По аналогии с временной сложностью, определяют пространственную сложность алгоритма, только здесь говорят не о количестве элементарных операций, а об объёме используемой памяти.

При оценке не учитывается место, которое занимает исходный массив и независящие от входной последовательности затраты, например, на хранение кода программы.

Асимптотические обозначения

Обозначение	**Интуитивное объяснение**	**Определение**
$f(n) \in O(g($	*f* ограничена сверху функцией*g* (с точностью до постоянного множителя) асимптотически асимптотическая верхняя граница	$\exists(C>0), n_0 : \forall(n>n_0)\ \lvert f(n)\rvert \leq \lvert Cg(n)\rvert$ или $\exists(C>0), n_0 : \forall(n>n_0)\ f(n) \leq Cg(n)$
$f(n) \in \Omega(g($	*f* ограничена снизу функцией *g*(с точностью до постоянного множителя) асимптотически асимптотическая нижняя граница	$\exists(C>0), n_0 : \forall(n>n_0)\ \lvert Cg(n)\rvert \leq \lvert f(n)\rvert$
$f(n) \in \Theta(g(n$	*f* ограничена снизу и сверху функцией *g* асимптотически асимптотические верхняя и нижняя границы	$\exists(C, C'>0), n_0 : \forall(n>n_0)\ \lvert Cg(n)\rvert < \lvert f(n)\rvert < \lvert C'g(n)\rvert$
$f(n) \in o(g(n))$	*g* доминирует над *f*асимптотически	$\forall(C>0), \exists n_0 : \forall(n>n_0)\ \lvert f(n)\rvert < \lvert Cg(n)\rvert$
$f(n) \in \omega(g(n$	*f* доминирует над *g*асимптотически	$\forall(C>0), \exists n_0 : \forall(n>n_0)\ \lvert Cg(n)\rvert < \lvert f(n)\rvert$
$f(n) \sim g(n)$	*f* эквивалентна *g*асимптотически	$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 1$

Примеры

«пропылесосить ковер» требует время, линейно зависящее от его площади (Θ(A)), то есть на ковер, площадь которого больше в два раза, уйдет в два раза больше времени. Соответственно, при увеличении площади ковра в сто тысяч раз, объем работы увеличивается строго пропорционально в сто тысяч раз, и т. п.

«найти имя в телефонной книге» требует всего лишь время, логарифмически зависящее от количества записей (O(log2(n))), так как открыв книгу примерно в середине, мы уменьшаем размер «оставшейся проблемы» вдвое (за счет сортировки имен по алфавиту). Таким образом, в книге, толщиной в 1000 страниц, любое имя находится не больше чем за раз (открываний книги).

Примеры сложностей алгоритмов

Название	**Сложность**	**Пример**	**Пример алгоритмов**
Константное время	O(1)	10	Определение четности числа Взятие элемента массива/хэш
Логарифмическое время	O(log n)	Log n, log n^2	Бинарный поиск
Линейное время	O(n)	N	Поиск наименьшего значение в неотсортированном массиве
Linearithmic time	O(nlogn)		Наиболее быстрый сортировки сравнением(quicksort и др)
Квадратичное время	O(n^2)	n^2	Сортировка пузырьком, сортировка вставками
Кубическое время	O(n^3)	n^3	Перемножение матриц nxn (без спец. алгоритмов)
Экспоненциальное время	2^O(n)	1.1^n, 10^n	Решение задачи о коммивояжере, динамическое программирование

Сортировка пузырьком.

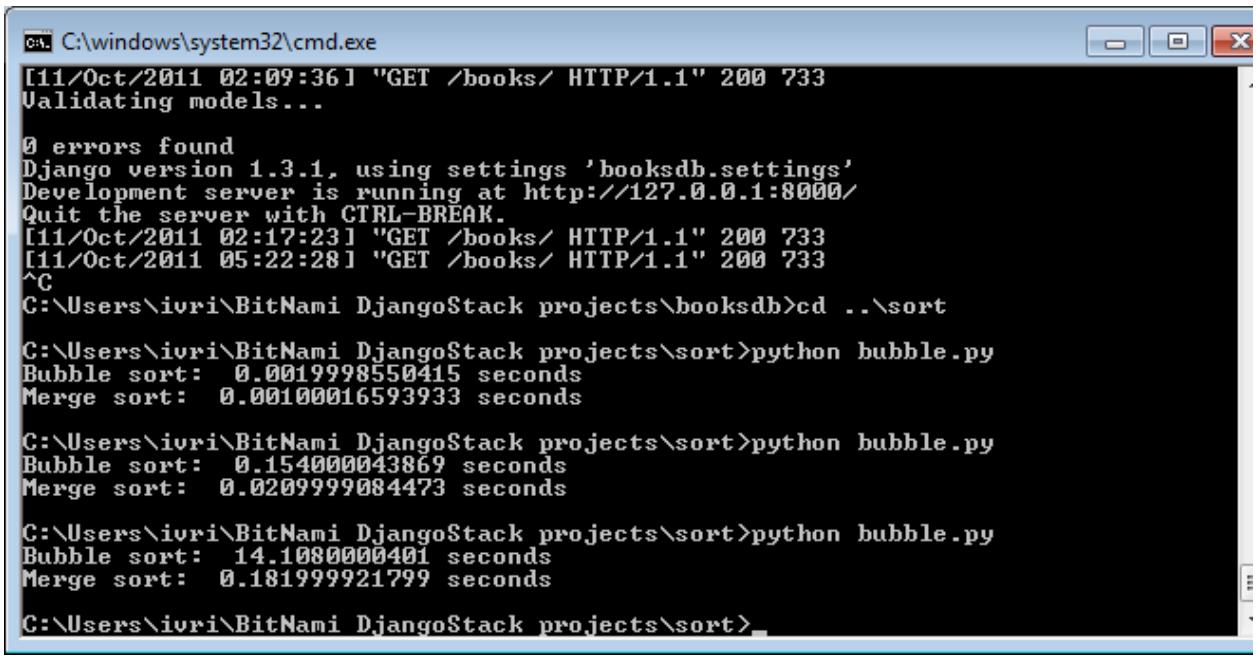
Временная сложность O(n^2):

```
def swap(arr, i, j):
        arr[i], arr[j] = arr[j], arr[i]
def bubble_sort(arr):
        i = len(arr)
        while i > 1:
                for j in xrange(i - 1):
                        if arr[j] > arr[j + 1]:
                                swap(arr, j, j + 1)
                i -= 1
```

Сортировка слияниями

Временная сложность O(n log n):

```
def mergesort(w):
 """Sort list w and return it."""
        if len(w)<2:
                return w
         else: mid=len(w)//2
# sort the two halves of list w recursively with mergesort and merge them
 return merge(mergesort(w[:mid]), mergesort(w[mid:]))
def merge(u, v):
"""Merge two sorted lists u and v together. Return the merged list r."""
        r=[]
        while u and v:
        # pop the smaller element from the front of u or v and append it to list r
                r.append( u.pop(0) if u[0] < v[0] else v.pop(0) )
        # extend result with the remaining end
                r.extend(u or v)
        return r
```



Выше приведено сравнение времени работы алгоритмов при сортировки 50, 500 и 5000 элементов (целочисленных) соответственно.

Код программы в отдельном файле bubble.py.

Лекция 6. Открытые системы. Сервис-ориентированная архитектура.

Аннотация.

Открытые информационные системы

Одной из главных тенденций современной индустрии информатики является создание *открытых систем.*

Свойство открытости означает:

1) переносимость (мобильность) ПО на различные аппаратные платформы,
2) приспособленность системы к ее модификациям (модифицируемость или собственно открытость)
3) приспособленность системы к комплексированию с другими системами в целях расширения ее функциональных возможностей и (или) придания системе новых качеств (интегрируемость).

Переход к открытым информационным системам позволяет существенно ускорить построение ИС в результате замены длительной и дорогостоящей разработки новых систем по полному циклу их компоновкой из ранее спроектированных подсистем или быстрой модернизацией уже существующих систем (реинжиниринг).

Открытость подразумевает выделение в системе интерфейсной части (входов и выходов), обеспечивающей сопряжение с другими системами или подсистемами, причем для комплексирования достаточно располагать сведениями только об интерфейсных частях сопрягаемых объектов. Если же интерфейсные части выполнены в соответствии с заранее оговоренными правилами и соглашениями, которых должны придерживаться все создатели открытых систем определенного приложения, то проблема создания новых сложных систем значительно упрощается. Из этого следует, что основой создания открытых систем являются стандартизация и унификация в области информационных технологий.

Значительное развитие концепция открытости получила в области построения вычислительных сетей, что нашло отражение в эталонной модели взаимосвязи открытых систем, поддерживаемой рядом международных стандартов. Идеи открытости широко используются при построении программного, информационного и лингвистического обеспечений ИС; в результате повышается степень универсальности программ и расширяются возможности их адаптации к конкретным условиям.

Аспекты открытости отражены в стандартизации:

- *API {Application Program Interface)* - интерфейсов прикладных программ с операционным окружением, в том числе системных вызовов и утилит операционной системы (ОС), т.е. связей с ОС;
- межпрограммного интерфейса, включая языки программирования;
- сетевого взаимодействия;
- пользовательского интерфейса, в том числе средств графического взаимодействия пользователя с ЭВМ;
- средств зашиты информации.

Стандарты, обеспечивающие открытость ПО, в настоящее время разрабатываются такими организациями, как ISO (International Standard Organization), IEEE (Institute of Electrical and Electronics Engineers), EIA (Electronics Industries Association) и др.

Среди других стандартов, способствующих открытости ПО ИС, следует отметить стандарты графического пользовательского интерфейса, хранения и передачи графических данных, построения баз данных и файловых систем, сопровождения и управления конфигурацией программных систем и др.

Важное значение для создания открытых систем имеют унификация и стандартизация средств межпрограммного интерфейса, или, другими словами, необходимо наличие профилей ИС для информационного взаимодействия программ, входящих в ИС. *Профилем* открытой системы называют совокупность стандартов и других нормативных документов, обеспечивающих выполнение системой заданных функций.

Так, в профилях ИС могут фигурировать унифицированный язык SQL обмена данными между различными СУБД, стандарты сетевого взаимодействия и т. п.

Мабрук – Краткие основы SOA *(http://www.ibm.com/developerworks/ru/edu/ws-soa-ibmcertified/)*

Введение в SOA

SOA - это архитектурный подход к определению, связыванию и интеграции повторно используемых бизнес-сервисов, имеющих четкие границы и самодостаточных по своей функциональности. В рамках такой архитектуры можно организовывать бизнес-сервисы в бизнес-процессы. Внедряя концепцию сервисов (более высокого уровня абстракции, не зависящего от приложений и платформы информационной инфраструктуры, а также от контекста или других сервисов), SOA переносит информационные технологии на следующий уровень, более подходящий для обеспечения функциональной совместимости и реализации в гетерогенных средах.

Поскольку SOA основывается на стандартах (таких, например, как Web-сервисы), утвержденных и поддерживаемых основными поставщиками информационных технологий, сервисы можно быстро создавать и объединять. Можно организовать взаимодействие предприятий независимо от их инфраструктуры, что обеспечивает делегирование, совместное применение, повторное использование и максимальную эффективность существующих активов.

При внедрении SOA вы переводите внутреннюю информационную инфраструктуру на более высокий, более открытый и управляемый уровень. С появлением повторно используемых сервисов и высокоуровневых процессов внесение изменений становится проще, чем когда бы то ни было, и начинает больше походить на разборку и сборку частей (сервисов) в новые процессы, направленные на осуществление бизнес-деятельности. Это не только способствует повышению эффективности и повторному использованию, но и дает возможность изменять и подстраивать информационные технологии под меняющиеся бизнес-требования.

Что лучше всего подходит для SOA?

Хотите узнать, для каких бизнес-функций и ситуаций архитектура SOA подходит лучше всего и где она может полностью проявить свой потенциал? Существуют определенные ситуации и бизнес-функции, когда следует немедленно обратиться к SOA, поскольку эта архитектура может существенно повысить конкурентоспособность и производительность и четко проявить свои преимущества. К таким ситуациям главным образом относятся:

- **Централизованные бизнес-функции, используемые несколькими субъектами**. SOA помогает идентифицировать эти функции и собрать их в повторно

используемые самодостаточные сервисы, не подверженные влиянию изменений в процессах, их использующих.

- **Интеграция с партнерами**. SOA способствует применению стандартов, создающих единые критерии для работы всех заинтересованных сторон. Кроме того, обеспечиваемая архитектурой SOA гибкость улучшает процесс интеграции благодаря возможности подключать, изменять и обновлять сервисы практически незаметно для ваших клиентов.

- **Наличие работающих старых технологий**. Некоторые организации не желают отказываться от проверенных и надежных старых технологий. Вопросы безопасности делают некоторых пользователей, особенно в сфере банковского обслуживания, недоверчивыми к новым программным системам и их неисследованным уязвимостям. В таких ситуациях SOA может помочь облачить старые технологии в новые стандарты, отобразить их в основанной на стандартах среде и сделать пригодными для интеграции и повторного использования.

Чем обусловлена быстрая настройка под меняющиеся бизнес-требования?

Из-за неизбежности изменений единственным гарантом обеспечения непрерывности бизнес-деятельности является способность адаптироваться к изменениям и быть готовым к ним (подвижность бизнеса - agility). SOA обеспечивает возможность адаптации к бизнес-требованиям (что имеет решающее значение для будущего любой деятельности), благодаря следующим факторам:

Слабое связывание

1. Устраняет жесткие связи, препятствующие изменениям.
2. Меньше вложений в реализацию и больше в повторное использование.
3. Улучшает возможности удаленного доступа к оригинальным источникам информации, уменьшая задержки и зависимости.
4. Проекты по интеграции управляются бизнес-требованиями (то есть бизнес-деятельность является основной движущей силой).
5. Благодаря отображению и совместному использованию информации, слабое связывание позволяет компаниям извлекать в режиме реального времени больше данных об эффективности бизнес-деятельности.
6. Облегчает партнерам взаимодействие с вашей компанией.
7. Способствует продвижению и публикации ваших сервисов, облегчая клиентам обнаружение их и вашей компании.
8. Облегчает поиск новых партнеров и сервисов, помогая найти более подходящий под ваши требования сервис.

Повторное использование

1. Делает процессы более согласованными, поскольку они базируются на одних и тех же компонентах.
2. Способствует повышению качества благодаря конкуренции между провайдерами сервисов.

3. Позволяет изменять систему независимо от изменений бизнес-деятельности.
4. Уменьшает влияние изменений, поскольку они выполняются централизовано и охватывают все участвующие стороны.

Расширяемость

1. Делает SOA-решения доступными организациям любого размера.
2. Изменяет процесс разработки на более динамичный, более подходящий для ведения бизнес-деятельности.
3. Ускоряет слияния и поглощения.

Так что же теряет компания, отказываясь от внедрения SOA?

Если SOA является приемлемым решением для компании, отказ от ее реализации может привести к следующим трем главным негативным последствиям:

- Неспособность охватить более крупные рыночные ниши, обеспечивающие дальнейшее развитие бизнес-деятельности. Поскольку компания привязана к своим специализированным системам, она застревает в своей первоначальной рыночной нише и затрачивает много усилий, чтобы пробиться выше. Напротив, используя SOA, организация может менять бизнес-тактики и реализовать новые, оставаясь на гребне.

В каких случаях применение SOA не обосновано

Архитектура SOA приносит пользу бизнес-организациям практически во всех ситуациях. Однако в очень специализированных случаях она может помешать улучшению бизнес-деятельности. К таким ситуациям относятся:

- **Когда информационная среда гомогенна**. Если организация использует комплекс согласованных продуктов (принадлежащих, например, одному производителю), SOA может оказаться помехой, а не полезной стратегией.
- **Когда критична производительность в режиме реального времени**. В силу слабого связывания между различными потребителями и производителями архитектура SOA зависит от протоколов взаимодействия, которые по своей природе являются медленными. Она также склонна применять логику посредничества и асинхронные протоколы, которые не подходят для эффективной работы в режиме реального времени.
- **Когда ничего не меняется**. Если потребитель не видит изменений в бизнес-логике, представлении, потоке данных, процессе или любых других аспектах приложения, преобразование старых систем в SOA может не оправдать затраченных усилий.
- **Когда тесное связывание не является недостатком**. Слабое связывание приносит пользу, когда оно используется с компонентом, который вами не управляется и изменения которого вы, следовательно, не можете контролировать. С другой стороны, когда компонент ваш и находится под вашим контролем, слабое связывание может потребовать дополнительных накладных расходов, особенно если компонент не является повторно используемым.

Концепции SOA

Давайте рассмотрим некоторые концепции архитектуры SOA, чтобы лучше понять, что она собой представляет.

Определение сервиса в SOA

Существует множество различных определений сервисов, но, на мой взгляд, лучше всего объясняют сущность сервисов определения, приведенные ниже.

Цитата из статьи "*Web-сервисы и сервис-ориентированная архитектура. Руководство продвинутого менеджера*" (ссылка приведена в разделе "Ресурсы"):

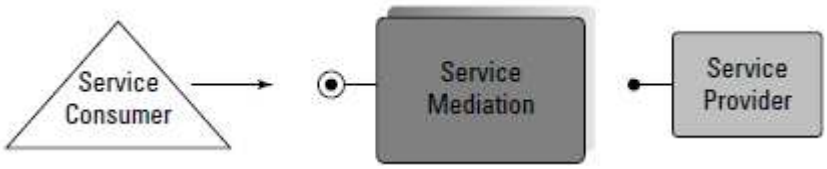
Сервис - это функция, являющаяся четко определенной, самодостаточной и не зависящей от контекста или состояния других сервисов.

Концепция слабого связывания в SOA

SOA Adoption for Dummies, 29 - 31

Виртуализация сервиса(Service Mediation).

Разделение сервиса происходит с помощью механизма виртуализации. Виртуальный сервис является прокси-объектом для реального сервиса. Прокси-сервис представляет собой желаемый потребителем сервиса интерфейс. Потребители обращаются к прокси-сервису, передающему сообщения к действительному сервису.



Виртуализация сервиса обеспечивает гибкость, необходимую при внедрении SOA. Эта гибкость основана на факте, что виртуальный сервис разделяет поставщика и потребителя в терминах местоположения, передачи данных и сообщений.

Независимость местоположения.

Виртуальный сервис позволяет скрыть действительное местоположение сервиса от потребителей. Это дает свободу перемещать реализацию сервиса без уведомления потребителей. Например, вы можете переместить сервис на сервера большей мощности для увеличения производительности.

Независимость передачи данных.

Виртуализация сервиса позволяет снабжать сервис несколькими средствами передачи данных. Предположим, вы создали сервис «CreateOrder», доступный через JMS (Java Message Service). Сервис стал популярен и некоторые пользователи желают расширить функциональность своих приложений данным сервисом. Сложность в том, что они могут использовать HTTP-протокол. Обычно требуется создать другую реализацию сервиса “CreateOrder” для поддержки HTTP, но возможности виртуальных сервисов позволяет создать виртуальный HTTP-сервис без изменения реализации. Это прозрачно решает проблему взаимодействия и позволяет расширять число пользователей сервиса.

Независимость сообщений.

Иногда потребители сервиса не синхронизированы с поставщиками в смысле ожидаемых сервисом XML-сообщений. В таких ситуациях виртуализация сервиса

предлагает трансформировать сообщения между форматами поставщика и потребителя. Подобный эффект может быть получен, например, при введении в эксплуатацию новой версии сервиса и изменении XML-схем, определяющих параметры сообщений. Предполагается, что потребители сервиса должны всегда соблюдать ожидаемый поставщиком формат. Но, при изменениях, сложно заставить всех потребителей мгновенно приспособиться.

Типовые функции виртуального сервиса

Виртуальный сервис – наилучшее место реализации некоторых технических условий или обеспечения качества сервиса (QualityOfService):

1. Проверка XML сообщений на корректность формата и соответствие интерфейсу сервиса.
2. Аутентификация и авторизация: идентификация потребителя сервиса и проверка наличия у него прав для вызова сервиса.
3. Расшифровка сообщений и проверка подписи.
4. Балансировка нагрузки и гарантии наличия ресурсов для работы сервиса.
5. Маршрутизация сообщений. Передача сообщений различным реализациям сервиса в зависимости от содержимого сообщений или внешних условий.
6. Мониторинг работы сервиса, производительности, а также проверка предоставления поставщикам требуемых услуг (SLA).

Данные требования изменяются много чаще, чем функциональная логика сервисов.

Механизм виртуализации позволяет реализовать разное качество сервиса (QoS) для разных клиентов, например, HTTP-аутентификацию внутри предприятия и передачу шифрованного XML для внешних клиентов.

Мабрук – Краткие основы SOA (*http://www.ibm.com/developerworks/ru/edu/ws-soa-ibmcertified/)*

Чтобы понять концепцию слабого связывания в SOA, следует прежде всего рассмотреть концепцию слабого связывания (loose coupling) в целом. Следующие утверждения демонстрируют, что такое слабое связывание и каково его значение:

- Сущность связывается, если ее изменение одной взаимодействующей стороной требует изменений другими сторонами (например, модели данных).

- Сущность объявляется, если ее поведение определяется в интерфейсе к сервису, а инициаторы запросов и провайдеры могут взаимодействовать только при совпадении объявленного поведения. К объявляемым аспектам относятся система защиты, транзакционное поведение и качество сервиса (например, время реакции и доставки).

- Сущность преобразовывается, если она объявляется как инициаторами запросов, так и провайдерами сервисов, но инициаторами и провайдерами заявлено различное поведение сущности и инфраструктура обеспечивает определенные возможности преобразования, позволяющие наладить взаимодействие.

- Сущность согласовывается, если как инициатор запросов, так и провайдер сервиса объявляют спектр поведений, которые они могут поддерживать, а посредническая инфраструктура может договориться о согласовании их поведения для каждого взаимодействия.

- Сущность развязывается, если изменения аспекта одной взаимодействующей стороной не требуют соответствующих изменений другими сторонами.

Слабое связывание проявляет себя в парадигме SOA следующим образом:

- Оно помогает организовать уровень абстракции между производителями и потребителями сервисов.
- Оно способствует реализации гибкости в изменении реализации сервисов без воздействия на потребителей сервисов.
- В архитектуре SOA функциональность организуется как набор модульных повторно используемых общих сервисов. Эти сервисы имеют четко определенные интерфейсы, инкапсулирующие ключевые правила доступа к ним. Они также строятся без каких-либо допущений о том, кто будет использовать или потреблять эти сервисы. Таким образом, они слабо связаны с потребителями сервисов.

Источники

1. Норенков – Основы автоматизированного проектирования 34-36.
2. Мабрук – Краткие основы SOA (http://www.ibm.com/developerworks/ru/edu/ws-soa-ibmcertified/). Секция 2-4 (Секция 4 до «XML в SOA»):
   http://www.ibm.com/developerworks/ru/edu/ws-soa-ibmcertified/section2.html
   http://www.ibm.com/developerworks/ru/edu/ws-soa-ibmcertified/section3.html
   http://www.ibm.com/developerworks/ru/edu/ws-soa-ibmcertified/section4.html
3. SOA Adoption for Dummies (Англоязычный), 29 – 31.

Лекция 7. Сервис-ориентированная архитектура.

Мабрук – Краткие основы SOA (http://www.ibm.com/developerworks/ru/edu/ws-soa-ibmcertified/)

XML и SOA

Архитектура SOA основывается на открытых стандартах и поддерживает платформенно-независимую бизнес-интеграцию, но она нуждается в общей технологии представления данных, на которой будет базироваться ее инфраструктура. Эта инфраструктура должна поддерживаться всеми участвующими сторонами и, чтобы служить основой для взаимопонимания. В центре этой инфраструктуры находится технология XML. Тому есть целый ряд причин:

- XML является фундаментом практически всех стандартов web-сервисов, в том числе XML Schema, SOAP, WSDL (Web Services Description Language) и UDDI (Universal Description, Discovery, and Integration). Эти стандарты опираются на основополагающую концепцию основанных на XML представлений - поддерживаемый во всем мире формат обмена информацией между провайдерами сервисов и инициаторами запросов в SOA.
- Использование XML решает проблему работы с различными форматами данных в различных приложениях, работающих на разных платформах.
- Преимущество XML заключается в простоте представления, являющегося по своей природе текстовым, гибким и расширяемым.

Примеры стандартов, основанных на XML и используемых в SOA:

- **SOAP**. Этот простой основанный на XML протокол позволяет приложениям обмениваться информацией по транспортным протоколам, таким как HTTP. Благодаря использованию XML протокол SOAP является:
    - Платформенно-независимым.
    - Пригодным для использования в Интернете.
    - Читабельным, структурированным и текстовым.

Благодаря всем этим преимуществам SOAP является рекомендованным и самым широко используемым коммуникационным протоколом для web-сервисов. А так как web-сервисы являются краеугольным камнем архитектуры SOA, этот протокол является также основным коммуникационным протоколом для основанных на SOA решениях.

- **WSDL**. Это документ, написанный на XML и описывающий web-сервис. Он определяет месторасположение сервиса и отображаемые им операции (или методы), позволяющие обращаться к этому сервису. WSDL-файл описывает четыре главные вещи:
    - Сервисы, доступные через интерфейс web-сервиса, такие как список имен методов и сообщений-атрибутов.
    - Тип данных сообщений.
    - Адрес сервиса, используемый для его вызова.

Реестры сервисов

Реестр сервисов представляет собой каталог сервисов, доступных в системе SOA. Он содержит физическое месторасположение сервисов, версии и их срок действия, а также

документацию по сервисам. Реестр сервисов является одним из основных строительных блоков архитектуры SOA. Его роль описывается ниже:

- Реестр сервисов реализует SOA слабое связывание. Храня месторасположения оконечных точек сервисов, он устраняет тесное связывание, приводящее к жесткой привязке потребителя к провайдеру. Он также облегчает потенциальные сложности замены одной реализации сервиса другой при необходимости.

- Реестр сервисов позволяет системным аналитикам исследовать корпоративный портфель бизнес-сервисов. Исходя из этого, они могут определить, какие сервисы доступны для автоматизации процессов с целью удовлетворения актуальных бизнес-потребностей, а какие нет. Это в свою очередь позволяет узнать, что нужно реализовать и добавить в портфель, формируя каталог доступных сервисов.

- Реестр сервисов может выполнять функцию управления сервисами, обязывая подписывающиеся сервисы быть согласованными. Это помогает гарантировать целостность руководства (governance) сервисами и стратегий. Дополнительная информация о руководстве и важности SOA приводится далее в данном руководстве.

Что такое бизнес-процесс?

Термин *бизнес-процесс* часто употребляется в среде SOA. Ниже приведены два определения бизнес-процесса:

Цитата из статьи "Бизнес-процессы и поток работ в мире web-сервисов" (developerWorks, январь 2003):

"Бизнес-процесс модет быть определен как набор взаимосвязанных задач, относящихся к деятельности, имеющей функциональные границы. Бизнес-процессы имеют начальные и конечные точки и являются повторяемыми ".

Примером бизнес-процесса является выдача служебного пропуска. Для инициации процесса вы предоставляете свидетельство о рождении, резюме, а также фотографию. Затем заводится личное дело, проводится проверка и после этого выдается пропуск.

В парадигме SOA бизнес-процесс управляет потоком сервисов. Бизнес-процесс управляет потоком событий, вызывает и координирует сервисы и создает контекст для их взаимодействия. Бизнес-процесс, будучи отделенным от реализации сервисов, заботится о ходе деятельности. Такое разделение задач позволяет больше сконцентрироваться на создании процесса и облегчает изменение процесса в соответствии с новыми требованиями.

Элементы бизнес-процесса

Возможно, лучше определить бизнес-процесс в понятиях составляющих его элементов; это позволяет взглянуть на бизнес-процесс с технической точки зрения.

- **Входные данные (input)** - информация, необходимая процессу для формирования результата. В примере с пропуском входными данными могут быть ваши резюме, свидетельство о рождении и фотография.

- **Выходные данные (output)** - все данные и информация, сгенерированные процессом. Выходные данные представляют собой бизнес-цели и показатели, необходимые для бизнес-деятельности. В примере с пропуском выходными данными могли бы быть личное дело и готовый пропуск, а также показатели работы процесса.

- **События (events)** - уведомления о возникновении чего-либо важного, например, визуальная индикация. Они могут возникать до, во время и после выполнения процесса. В примере с пропуском событием могло бы быть предоставление нового документа, который ранее отсутствовал и который должен быть включен в личное дело.

- **Подпроцесс (subprocess**) - более мелкий процесс или этап в рамках процесса. Подпроцесс используется тогда, когда невозможно представить объем работы одним набором действий. Он имеет те же элементы, что и процесс. В примере с пропуском подпроцессом могло бы быть исследование вашего досье и получение результатов.

- **Действие (activity)** - наименьший элемент работы в процессе. В нашем примере действием могло бы быть создание нового личного дела для человека, получающего пропуск.

- **Показатели производительности (performance metrics**) - атрибуты, представляющие эффективность процесса для определения его соответствия необходимой производительности. Эти показатели помогают определить производительность и сравнить ее с требующимися значениями. Они также выделяют потенциальные области совершенствования процесса, реализуя в конечном итоге цикл улучшений, обещанный архитектурой SOA. В примере с пропуском эти показатели могли бы определять, выполнение какой части процесса занимает больше всего времени либо приводит к достижению пика загрузки. Они помогают дальнейшему совершенствованию процесса.

Как SOA управляет транзакциями?

Поскольку в процесс вовлечено множество действий, бизнес-транзакции, возникающие в среде SOA, могут быть очень сложными. Причиной этого является природа сервисов долго выполняющихся процессов в контексте SOA, которые часто являются асинхронными, не сохраняющими состояния, распределенными и непрозрачными.

Web-сервисы являются отличным представлением сервисов в среде SOA. Будучи самодостаточными (согласно требованиям SOA), они являются ограниченными с точки зрения межсервисных транзакций. Поскольку сервис находится на вершине транзакции, а область действия транзакции ограничена действиями, выполняемыми логикой сервиса, нет необходимости в реализации функциональности межсервисных транзакций, а сами транзакции могут управляться любой закрытой технологией (компонентной, традиционной или какой-либо другой), которая инкапсулирована в сервисе. Но с ростом числа сервисов увеличивается необходимость распространения транзакций на несколько сервисов.

Для решения проблемы транзакций был разработан ряд спецификаций web-сервисов. К ним относятся:

- **WS-Coordination**. Позволяет зарегистрированным процессам принимать участие в создании общего контекста, ответственного за хранение текущих данных и распространяемой между ними информации. Координация обработки существующих транзакций, потоков работ и других систем осуществляется интегрированной средой. Это позволяет скрыть проприетарные протоколы и работать в гетерогенной среде. Этот протокол обеспечивает инфраструктуру для других протоколов, таких как WS-AtomicTransaction или WS-BusinessActivity.

- **WS-AtomicTransaction**. Используется в краткосрочных распределенных действиях. Предоставляет три типа протоколов, которые могут использоваться с интегрированной средой WS-Coordination для реализации транзакций с двухфазной фиксацией типа ACID (транзакций, поддерживающих атомарность, согласованность, изоляцию и устойчивость).

- **WS-BusinessActivity**. Этот протокол используется с долго работающими транзакциями.

Основы архитектуры SOA

Теперь рассмотрим некоторые более сложные технические аспекты, такие как роль корпоративной сервисной шины (enterprise service bus - ESB), бизнес-процессы и их хореография (choreography), а также роль web-сервисов.

Составляющие базовой архитектуры SOA

Базовая архитектура SOA состоит из провайдера сервисов, сервиса и (необязательного) каталога сервисов. Для обмена информацией используется механизм обмена сообщениями типа "приложение к приложению".

Сходство между этой моделью и чистыми web-сервисами совершенно очевидно, поскольку в обоих случаях применяется WSDL-документ, являющийся контрактом по активизации, хранящимся в каталоге сервисов, из которого этот сервис может быть запрошен и извлечен посредством механизма UDDI. web-сервисы в действительности являются реализацией архитектуры SOA на самом базовом уровне.

В этой модели базовый сценарий таков. Сначала провайдер сервиса создает сервис, принимает решение открыть этот сервис и публикует его. Публикация выполняется путем отправки информации о сервисе в каталог сервисов. С другой стороны, инициатор запросов сервиса (service requester), нуждаясь в определенном сервисе, просматривает каталог сервисов в поисках того из них, который удовлетворяет необходимому критерию. После обнаружения такого сервиса и использования доступной в каталоге сервисов информации инициатор запросов сервиса может напрямую обратиться к провайдеру сервисов надлежащим способом для удовлетворения бизнес-потребности.

Рисунок 1. Базовая архитектура SOA

Ниже приведены определения некоторых понятий, используемых в данном разделе:

- **Провайдер сервиса**. Предоставляет сервисы, контракт по активизации которых и месторасположение опубликованы.

- **Потребитель сервиса**. Потребляет сервисы, соответствующие его бизнес-потребностям и обнаруженные в каталоге сервисов.

- **Каталог сервисов**. Служит для публикации и ведения списка сервисов, доступных для потребителей.

Роль ESB в архитектуре SOA

Wikipedia – ESB.

Enterprise Service Bus (сервисная шина предприятия) — подход к построению распределённых корпоративных информационных систем. Обычно включает в себя промежуточное ПО, которое обеспечивает взаимосвязь между различными приложениями по различным протоколам взаимодействия.

Существует некоторое разногласие, что именно считать ESB — архитектуру или программное обеспечение. Обе точки зрения имеют право на существование.

Архитектура ESB заключается во взаимодействии всех приложений через единую точку, которая, при необходимости, обеспечивает транзакции, преобразование данных, сохранность обращений. Данный подход обеспечивает большую гибкость, простоту масштабирования и переноса. При замене одного приложения подключенного к шине нет необходимости перенастраивать остальные.

Конкретные реализации ESB содержат в себе адаптеры для соединения с другим ПО.

Среди популярных можно назвать SAP NetWeaver XI/PI (Exchange Infrastructure/Process Integration) от SAP AG, BizTalk от Microsoft, WebSphere от IBM, JBoss — опенсорсный продукт, поддерживаемый RedHat.

На EclipseCon 2009 было объявлено о выходе первой версии Eclipse Swordfish ESB.

Мабрук – Краткие основы SOA (http://www.ibm.com/developerworks/ru/edu/ws-soa-ibmcertified/)

ESB играет важную роль в архитектуре SOA. По сути, она предоставляет магистральную сеть и инфраструктуру для соединения провайдеров и потребителей сервисов.

Роли ESB в информационной системе:

1. Предоставляет интеграционную инфраструктуру, соответствующую принципам SOA:
   1.1. Устанавливает явные независимые от реализации интерфейсы для организации слабого связывания.
   1.2. Использует коммуникационные протоколы, независимые от расположения взаимодействующих сторон.
   1.3. Способствует определению сервисов, инкапсулирующих повторно используемые бизнес-функции.
2. Предоставляет средства для управления инфраструктурой сервисов.
3. Функционирует в распределенной гетерогенной среде через поддержку синхронных и асинхронных взаимодействий, а также использование стандартных интерфейсов.
4. Централизует управление и распределяет обработку.
5. Реализует защиту и обеспечение качества сервиса в проектах SOA.

Недостатки ESB:

- Требует достаточно больших трудозатрат и специфических знаний для реализации, при этом сама по себе (без дальнейшей реализации SOA) практически не приносит ощутимой пользы для бизнеса;
- По сравнению с простейшей (точка-точка) интеграцией между системами, вносит задержки, связанные с преобразованием XML сообщений.
- Требует тщательного продумывания и контроля над версионностью сообщений, в противном случае может увеличить связность систем друг с другом (при недостаточной унификации сообщений);

Оркестровка и хореография

Крис Пельтц – Оркестровка и хореография веб-сервисов.

ИТ организаций должны адаптироваться под требования клиентов и условия рынка. Существующие языки описания бизнес-процессов не поддерживают напрямую веб-сервисы, что заставляет организации разрабатывать собственные проприетарные протоколы компоновки сервисов. Стандарты оркестровки и хореографии позволяют решать эти задачи.

http://searchsoa.techtarget.com. SOA orchestration and choreography

Организация OMG определяет оркестровку как «моделирование направленных, внутренних бизнес-процессов», а хореографию как «спецификацию взаимодействий между автономными процессами».

Оркестровка в бизнес-процессах – это серия действий в управляемом потоке работ, обычно имеющем одну линию выполнения.

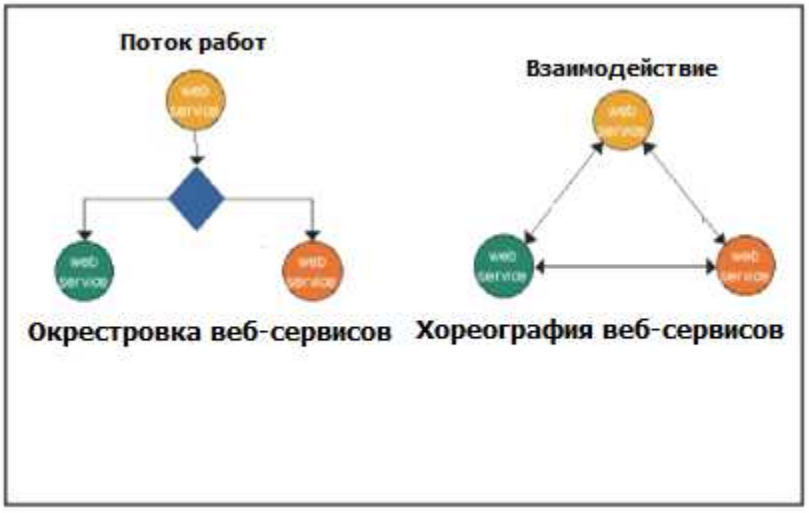
Хореография отражает видимый обмен сообщениями, правила взаимодействий и соглашения между двумя и более сервисами.

Ключевые элементы проектирования:

Для оркестровки: участник и его роль, переменные и свойства, определяющие взаимодействие участников, обработчики ошибок, события.

Для хореографии: структура сообщений, асинхронная и синхронная коммуникация сервисов, служебные сообщения.

Крис Пельтц – Оркестровка и хореография веб-сервисов.



Стандарты WSCI (Web Service Choreography Interface) и BPEL4WS(Business Process Execution Language for Web Services) разработаны для облегчения взаимодействия веб-сервисов.

Оркестровка отличается от хореографии тем, что она описывает процесс, протекающий между сервисами, контролируемый основным участником. В хореографии нет участника, ведущего обмен сообщениями.

Технические требования для оркестровки и хореографии.

Определим требования к оркестровке и хореографии как к инфраструктуре СОА:

1. ***Гибкость***. Достигается разделением между логикой процесса и веб-сервисами. Достигается реализацией логики процесса с помощью оркестровки.
2. ***Простые и структурированные действия***. Язык оркестровки должен поддерживать действия как для обращения к другим веб-сервисам, так и для описания семантики процесса. Простое действие можно рассматривать как компонент, взаимодействующий с чем-то вне процесса, в то время как структурированное действие управляет общим выполнением процесса, специфицируя состав и порядок действий.
3. ***Рекурсивная композиция***. Отдельный бизнес-процесс может взаимодействовать с множеством веб-сервисов. Сам процесс может быть представлен как веб-сервис, для агрегации в процесс более высокого уровня.

Дополнительно, оркестровка и хореография предъявляют требования к целостности и стабильности взаимодействий. Они включают:

1. Хранение состояний и корреляция запросов. Способность хранить состояние между запросами веб-сервисов особенно важно, когда работа ведется с асинхронными сервисами. Язык и инфраструктура должны обеспечивать хранение данных и корреляцию запросов для построения диалогов более высокого уровня.
2. Обработка исключений и транзакции. Долго выполняемые сервисы должны обеспечивать транзакционную целостность и управление исключениями.

WSCI

WSCI определяет расширение WSDL для взаимодействия сервисов. Первоначально составленный в Sun, SAP, BEA и Intalio, он стал спецификацией W3C. Это язык хореографии, описывающий обмен сообщениями между сервисами и не определяющий выполнение бизнес-процесса.

Один интерфейс описывает только сообщение единственного участника обмена. Хореография включает набор интерфейсов, по одному на каждого участника. Нет контроллера, регулирующего обмен.

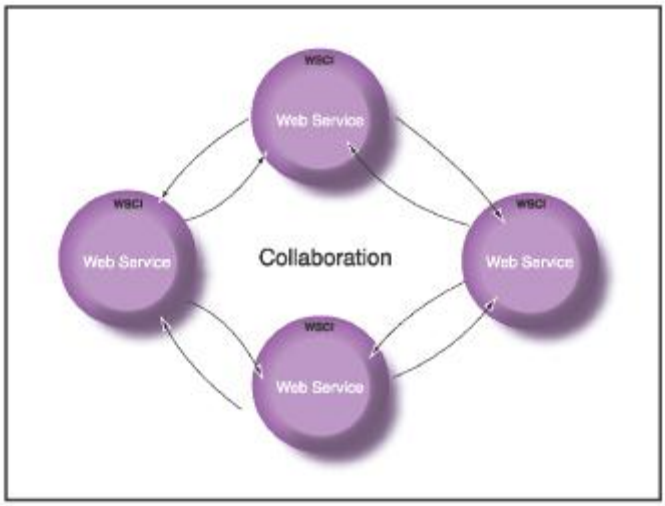


FIGURE 2 | **Web Services Choreography Interface (WSCI)**

Каждое взаимодействие является единицей работы, имеющей конкретную WSDL-спецификацию. WSDL описывает входные точки сервиса, WSCI описывает взаимодействие между WSDL-операциями.

WSCI может специфицировать запрос к сервису, внутреннему или внешнему.

BPEL4WS

BPEL4WS поддерживает как абстрактные бизнес-протоколы, так и выполняемые бизнес-процессы.

- ***Бизнес-протокол*** поддерживает публичный обмен сообщениями между участниками обмена. Его нельзя выполнить и он не определяет внутреннее выполнение процесса.
- ***Выполняемый процесс*** моделирует выполнение действий. Он обеспечивает оркестровку, в то время как бизнес-протоколы сфокусированы на хореографии.

Спецификация поддерживает простые действия для общения с веб-сервисами. Типичный сценарий заключается в приеме сообщения выполняемым процессом. Процесс запрашивает дополнительную информацию, вызывает внутренние

сервисы и возвращает результаты.

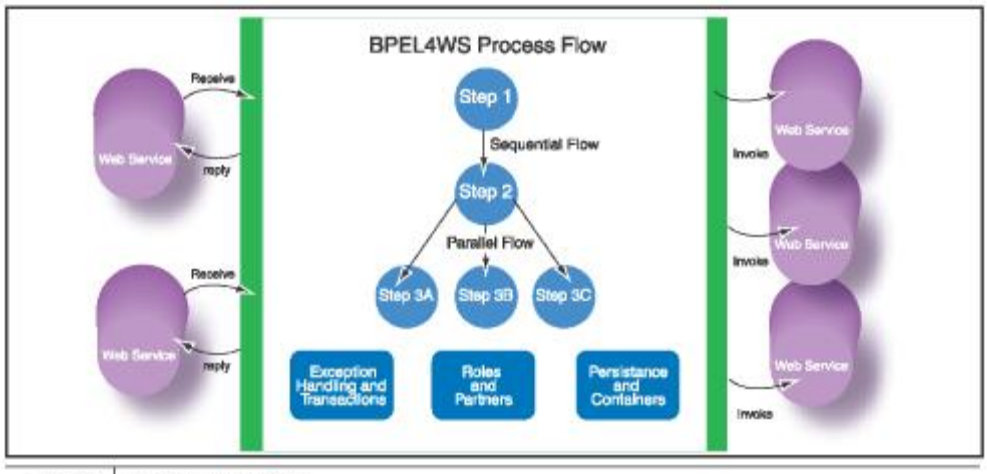


FIGURE 3 | **BPEL4WS Process Flow**

BPEL4WS поддерживает структурированные действия для построения бизнес-логики процесса. Переменные и партнеры – важные составляющие BPEL4WS

Мабрук – Краткие основы SOA (http://www.ibm.com/developerworks/ru/edu/ws-soa-ibmcertified/**)**

Процесс, определенный в BPEL4WS, состоит из:

- Действий (activities), которые являются отдельными бизнес-этапами внутри процесса. Действия могут быть простыми или состоять из других действий (структурированными).
- Ссылок на партнеров, которые определяют внешние сущности, взаимодействующие с процессом или, наоборот, использующие WSDL-интерфейсы.
- Переменных, хранящих сообщения, передаваемые между действиями, и, следовательно, представляющих состояние.
- Корреляционных наборов (correlation sets), использующихся для корреляции нескольких сообщений запросов сервиса и ответов с одним экземпляром бизнес-процесса. (К сервису могут обращаться различные бизнес-процессы, нельзя смешивать результаты работы для разных бизнес-процессов).
- Обработчиков неисправностей (fault handlers), занимающихся исключительными ситуациями, которые могут возникнуть во время работы бизнес-процесса.
- Обработчиков событий (event handlers), принимающих и обрабатывающих сообщения параллельно с обычным выполнением процесса.
- Корректирующих обработчиков (compensation handlers), определяющих логику коррекции для отката действия или нескольких действий при возникновении исключительной ситуации.

Лекция 8. Роль человека в СОА. Интеграция BPM, SOA, Web 2.0.

Роль человека в СОА

Задания для пользователей

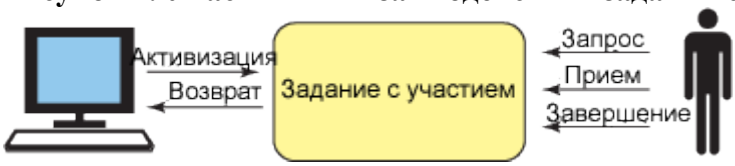
IBM – Краткие основы СОА.

Хореография обеспечивает также поддержку *заданий для пользователя (human tasks)*, включающими действия пользователей при взаимодействии либо с сервисом, либо с другим субъектом. Примером является административное утверждение заявки на командировку или обработка запроса клиента сотрудником компании.

Существуют следующие типы заданий для пользователей:

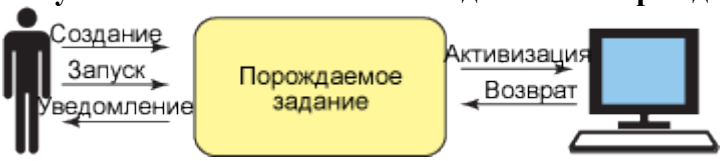
- **Задания с участием (participating tasks)**. Активизируются системой (процессом), требующей реакции пользователя для продолжения работы. Система инициирует задание, а субъект из списка кандидатов-исполнителей принимает и выполняет его. Затем он предоставляет ответ в систему, уведомляя о его завершении. Примером такого задания является процесс возмещения командировочных расходов, ожидающий утверждения менеджером.

Рисунок 2. Участники и взаимодействия задания с участием



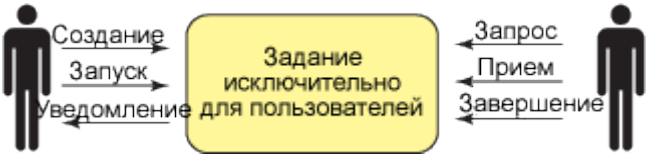
- **Порождаемые задания (originating tasks)**. Активизируются субъектом через пользовательский интерфейс. Они предназначены для системы: субъект создает порождаемое задание и запускает его; запрос передается системе для запуска требуемых сервисов. После завершения инициатору отправляется уведомление.

- **Рисунок 3. Участники и взаимодействия порождаемого задания**



- **Задания исключительно для пользователей (purely human tasks)**. Создаются и запускаются субъектом, предназначены для другого субъекта, который принимает и завершает их. Такие задания не взаимодействуют с бизнес-процессом или другими Web-сервисами. Они не являются автоматизированными и проходят тот же цикл назначения и уведомлений.

Рисунок 4. Участники и взаимодействия заданий исключительно для пользователей



Логично, что задания для пользователей длятся намного дольше, чем автоматизированные задания, что поднимает еще один вопрос: могут ли процессы позволить себе прерывание и время ожидания, вызываемые заданиями для пользователей?

Ответ - да. Для получения более подробного ответа давайте рассмотрим типы бизнес-процессов.

Типы бизнес-процессов

Бизнес-процессы могут быть либо долго выполняющимися процессами, либо микропотоками:

- **Долго выполняющиеся (long running) процессы** прерываемы и могут работать в нескольких транзакциях. Они могут ожидать внешних сигналов, аналогичных тем, которые являются результатом работы заданий для пользователей. Практическое правило заключается в том, что если процесс содержит задание для пользователя, он должен быть долго выполняющимся. Такие процессы могут содержать синхронные и асинхронные сервисы. Долго выполняющиеся процессы запоминают каждое промежуточное состояние процесса, того чтобы иметь возможность отката.

- **Микропотоки (micro-flows)** выполняются без прерывания за одну транзакцию, имеют малую длительность и состоят только из синхронных сервисов.

Роли в создании СОА.

SOA Adoption for Dummies, 53

В процесс жизненного цикла вовлечены:

1. ***Владелец бизнеса***: определяет требования, необходимые для создания новых деловых возможностей. Наилучший способ их выражения – в нотации моделирования бизнес-процессов (Business Process Modeling Notation, BPMN). Формальное описание процессов облегчает реализацию ИТ. Владелец бизнеса должен описать, помимо функциональности, нефункциональные характеристики, такие как качество предоставление услуг (QoS –Quality of Service).
2. ***Архитектор***: анализирует бизнес-требования и воплощает их в конструкции сервисов и бизнес-процессов. Он принимает решение: использовать имеющийся компонент, или создать новый. При создании нового или изменении существующего компонента, архитектор формирует проектные спецификации в форме диаграмм состояний, моделей процессов и интерфейсов. Архитектор формализует нефункциональные требования (доступность, безопасность, производительность и прочее).
3. ***Разработчик***: Внедряет компоненты, основанные на проектных спецификациях архитектора. Он создает план тестирования. Для реализации технологии разработчик использует заготовки, автоматически создаваемые на основе спецификаций: forward-engineering, reverse-engineering.
4. ***Управляющий качеством***: использует данные остальных участников, проверяет корректность реализации. Он использует план тестирования, оценивает качество, побочные эффекты, нефункциональные характеристики.
5. ***Оператор***: получает проверенное решение и применяет его в работе ИТ-системы. Он использует формализованные нефункциональные требования для обеспечения требуемого качества услуг.

Цикл жизни SOA и различные его этапы

Архитектура SOA характеризуется динамическим циклом жизни. Для нее свойственна возможность постоянного улучшения процессов, что, наряду с применяемой в SOA

концепцией слабого связывания, позволяет совершенствовать процессы через разборку и повторную сборку строительных блоков (сервисов в данном случае).

Диаграмма цикла жизни SOA содержит четыре смежных шестиугольника, представляющих четыре этапа SOA. Как видно на диаграмме, изображенной на рисунке (Рисунок 1), эти четыре этапа образуют непрерывный замкнутый цикл мониторинга и улучшения.

Рисунок 1. Четыре этапа цикла жизни SOA

Рассмотрим эти этапы более подробно.

Этап моделирования

Этап моделирования включает в себя анализ деятельности и сбор требований, за которыми следует моделирование и оптимизация бизнес-процесса. Модель помогает сформировать общее понимание процесса, его целей и результатов. Она также гарантирует соответствие проекта бизнес-требованиям и обеспечивает исходные критерии для последующего измерения производительности.

Этап сборки

На этом этапе существующие активы (например, ERP-сист, финансовые системы и т.д.), которые будут использоваться в моделируемых процессах, инкапсулируются в сервисы, тогда как отсутствующие нужные функциональности реализуются и тестируются. Когда все сервисы становятся доступными, можно скоординировать их для реализации бизнес-процесса.

Этап развертывания

На этапе развертывания среда времени исполнения настраивается на необходимый уровень качества сервиса и на удовлетворение требований безопасности. Среда может масштабироваться и оптимизироваться для повышения надежности работы критичных процессов и для обеспечения гибкости, позволяющей динамически обновлять ее в случае каких-либо изменений.

Этап управления

На этом этапе выполняется мониторинг ряда аспектов, таких как доступность сервисов, время реакции, контроль версий сервисов. Важную роль на этом этапе играет мониторинг ключевых показателей производительности (key performance indicators - KPI) процесса. Он помогает предотвратить или изолировать и диагностировать возникающие проблемы в реальном времени, а также обеспечить обратную связь с целью улучшения производительности бизнес-процесса и устранения узких мест. Эта обратная связь передается на этап моделирования (первый этап) с целью усовершенствования процесса.

Управление архитектурой SOA

Как отмечалось в первом разделе, SOA нуждается в интегрированной среде управления (management), в противном случае она выходит из-под контроля. Управление SOA реализуется в рамках концепции руководства (governance), контролирующей различные аспекты архитектуры SOA. Еще один аспект, который должен обязательно реализовываться в SOA-среде по причине ее открытой природы – система безопасности. Подробности механизма управления SOA рассматриваются в данном разделе.

Руководство SOA

IBM – Effective SOA governance

Руководство ИТ согласует действия в сфере ИТ с целями организации. Этот тип руководства включает принятие решений, связанных с ИТ-инвестициями, правилами и процессами, используемыми для измерения и контроля ИТ-решений.

Организации должны определить, какие операции могут быть реализованы сервисами и быть способными оценить функционирование сервисов.

Руководство СОА это расширение ИТ-руководства направленное на жизненный цикл сервисов и смешанных приложений. Внедрение руководства СОА часто начинает процесс совершенствования корпоративного и ИТ-руководства.

SOA Governance – Play of the Game, BY PHILLIP WINDLEY

Основная цель правил проектирования систем в усилении модульности и создании абстракций. Когда эти правила согласованы и утверждены, они называются политиками. Политики СОА в разработке и обеспечении функционирования называются руководством СОА.

Политики могут отражать любой аспект жизни сервиса, включая разработку, внедрение и выполнение. Например, политика разработки может утверждать корпоративное пространство имен, политика внедрения может требовать соблюдение сервисом ряда стандартов. Политика выполнения может требовать управления всех сервисов корпоративной системой безопасности.

IBM – Краткие основы СОА.

Руководство SOA (governance) - это интегрированная среда принятия решения и идентификации ролей для поддержки действий, согласующихся со стратегией предприятия, и предотвращения действий, не являющихся таковыми. Эта интегрированная среда управляется группой или комитетом, ответственным за создание стратегии руководства и идентификацию ролей, полномочий и ответственности индивидуумов, которым дается право принятия решений и проведения стратегии. Коротко говоря, комитет должен решить три основных вопроса:

- Какие решения необходимо выполнить, чтобы гарантировать эффективное управление информационными активами?
- Кто ответственен за принятие этих решений?
- Как можно привести эти решения в исполнение и проконтролировать это?

В рамках реализации механизма руководства идентифицируются и контролируются соглашения по уровню сервиса (service level agreements - SLA). Также собираются показатели производительности с целью определения эффективности руководства.

Какую роль играет руководство в среде SOA?

Руководство в SOA играет решающую роль; оно должно фигурировать во всех этапах цикла жизни SOA, как показано на рисунке 2.

Рисунок 2 Место руководства по отношению к этапам цикла жизни SOA

Необходимость руководства для SOA очевидна по следующим причинам:

- Руководство подразумевает применение принципов корпоративной стратегии для направления деятельности и управления информационными активами.
- Руководство гарантирует поддержание заданного уровня целостности, производительности, надежности и актуальности сервисов.
- Руководство гарантирует корректную оценку требований бизнес-приложения и расстановку приоритетов при создании и потреблении сервисов, способствуя, таким образом, наилучшему использованию в соответствии с бизнес-целями.
- Руководство гарантирует рентабельность инвестиций в информационные технологии.
- Руководство гарантирует, что корпоративная архитектура SOA является основным ориентиром при проектировании любого сервиса.
- Руководство, как контролирующий орган, опирается на передовые методики применения информационных технологий.
- Защищая бизнес-активы, руководство способствует также защите корпоративных данных и конфиденциальности информации, используемой в различных регионах.
- Руководство, управляя информационными технологиями предприятия, способствует целостности и надежности данных и процессов с целью получения преимуществ от повторного использования и максимизации прибыли.
- Руководство гарантирует требуемый уровень производительности и качества сервисов всех компонентов в цепочке потребитель-провайдер.

- Стандарты являются основой SOA, поэтому руководство гарантирует высокий уровень совместимости, открывающий предприятию все преимущества открытых стандартов.

- Руководство использует показатели для аудита и мониторинга продвижения разработки информационной инфраструктуры и ее соответствия утвержденной стратегии.

Соответствие качества сервиса установленным требованиям в руководстве SOA

В интегрированной среде с руководством SOA стратегии QoS определяются и реализуются на уровне предприятия. Это жизненно важно в открытой среде, где интеграция и обмен информацией между сервисами не ограничен внутренними функциями предприятия, а выходит за его пределы на предприятия различного размера, различных областей и с различными информационными технологиями, причем должен поддерживаться и гарантироваться устойчивый уровень общих процессов. Например, один не отвечающий требованиям сервис может поставить под угрозу всю систему. В некоторых системах инфраструктура способна определять уровни QoS и отклонять не отвечающие требованиям сервисы.

Почему системы безопасности в средах SOA являются сложными и распределенными?

Такие сложные системы безопасности необходимы по следующим причинам:

- Распределенные системы требуют распределенной системы безопасности.

- Существует необходимость управления реестрами пользователей и управления доступом для нескольких приложений, платформ, бизнес-партнеров и сущностей, чего нельзя сделать из единой точки.

- Необходимо последовательно осуществлять политики безопасности во всей среде.

- Система безопасности должна быть способна развиваться по мере развития предприятия и его приложений.

Влияние изменения в сервисах на цикл жизни SOA

Учитывая принцип слабой связи, изменения сервисов в среде SOA выполняются просто, поскольку потребители сервисов отделены от провайдеров сервисов посредством шины ESB, расположенной между ними и способной быть посредником при обмене сообщениями. Изменения на стороне провайдера могут учитываться в ESB, так чтобы потребитель оставался таким же.

С другой стороны, необходимо подчеркнуть важность неуправляемых изменений в среде SOA. Согласно принципу повторного использования, каждый сервис может быть сервисом корпоративного уровня, а не только локальным и используемым в одном отделе или подразделении. Любые неуправляемые изменения в таком сервисе могут привести к непредсказуемым сбоям на всем предприятии и к остановке процессов. Это демонстрирует важность руководства в реализации стратегии управления изменениями. Эта стратегия должна оценивать последствия, разрешать изменения и гарантировать реализацию системы уведомлений для затронутых изменением участников

взаимодействий (ESB или непосредственных пользователей). Управление изменениями в распределенных системах должно осуществляться на основании строгих правил.

Роль ESB в руководстве SOA

ESB играет важную роль в осуществлении руководства. К ESB могут быть применены стратегии системы безопасности и QoS с целью управления их уровнями и разрешения только допустимых запросов. В целом ESB играет роль унифицированной платформы, на которой реализуются необходимые стратегии. По своей природе ESB является центром, где происходят все взаимодействия, что делает ее идеальным местом для применения правил. Это, гарантирует, что каждый участник процесса либо соответствует этим правилам, либо изолируется.

Подготовка к реализации SOA

Процесс внедрения SOA в организации требует наличия определенной квалификации, включая:

- Способность определить готовность организации к такому внедрению.
- Информирование людей о преимуществах и недостатках СОА.
- Оценку проблем и стимулов для внедрения SOA как с коммерческой, так и с технической стороны.

Факторы внедрения SOA с точки зрения бизнеса

Для бизнеса важно знать форму и влияние, которые эта новая парадигма окажет на организацию, поэтому необходимо идентифицировать возможные проблемы.

Проблемы с точки зрения бизнеса

К таким проблемам относятся:

- Сомнения менеджмента из-за того, что SOA больше управляется ИТ, чем бизнесом.
- Отображение процессов на сервисы.
- Отсутствие знаний о SOA и о ее возможностях.
- Неправильное представление о том, что SOA является только архитектурным подходом к информационным технологиям, что может привести к пренебрежению критически важной ролью руководства.

В своем большинстве указанные проблемы могут быть решены, или, по крайней мере, идентифицированы, путем проведения учебных занятий с целью демонстрации преимуществ и реального значения SOA.

Побуждающие факторы

Основным побуждающим фактором является потенциал SOA для:

- Повышения окупаемости (return on investment - ROI) благодаря уменьшенной стоимости реализации за счет внедрения стандартов, повторного использования, отображения сервисов и интеграции с партнерами.
- Уменьшения времени выхода на рынок за счет повторного использования активов и внедрения сервисов, предоставляемых партнерами.

- Улучшения видимости информационных активов и их согласования с бизнес-целями.
- Повышения гибкости как внутренней (при взаимодействиях), так и внешней (при работе с партнерами).
- Реализации более эффективных процессов за счет повторного использования информационных активов и стандартов.
- Обеспечения гибкости бизнес-деятельности и способности легко и быстро адаптироваться к рыночным изменениям.
- Уменьшения затрат по организации.

Факторы внедрения SOA в организациях с точки зрения ИТ.

Некоторые сдерживающие и побуждающие факторы, важные для ИТ-службы, перечислены ниже.

Проблемы с точки зрения информационных технологий

К таким проблемам можно отнести:

- Преобразование существующих специализированных систем в основанные на стандартах сервисы.
- Управление, руководство и контроль сервисов.
- Проблемы защиты распределенных систем.
- Надежность новых систем по сравнению с существующими проверенными системами.
- Оптимизация и унификация существующих активов для устранения избыточности.

Побуждающие факторы с точки зрения информационных технологий

Такими факторами могут быть:

- Внедрение стандартов. Это также считается проблемой, но, несмотря на усилия, необходимые для их внедрения, преимущества очевидны для каждого ИТ-специалиста.
- Гарантирование высокого качества сервиса.
- Повторное использование существующих информационных активов.
- Слабое связывание сервисов.
- Независимость от конкретного провайдера или партнера.

Идентификация преград для внедрения SOA

Организации необходимо идентифицировать и попытаться преодолеть все преграды, блокирующие внедрение SOA. Вот примеры возможных преград:

- Консервативные ИТ-специалисты настаивают на старой модели разработки по принципу "водопада".

- Мнение о том, что сложные системы лучше, боязнь неизвестного.
- Игнорирование важности архитекторов и стремление считать их теоретиками, стоящими дороже, чем того требует решение. Следует отметить, что архитекторы играют важнейшую роль в SOA, и их отсутствие непременно приведет к нежелательным результатам.
- Организационное сопротивление внедрению модели SOA. SOA требует кооперации всех групп в организации, а не только простой реализации интегрированной информационной среды.

Интеграция BPM, SOA и WEB 2.0

BPM, SOA, Web 2.0 - Oracle Whitepaper

Управление бизнес-процессами(BPM) представляет собой стратегию руководства и улучшения деятельности предприятия через постоянный процесс оптимизации в замкнутом цикле моделирования, выполнения и измерения. В дополнение, использование средств управления ресурсами, управления связей с клиентами (CRM), решений business intelligence позволяет BPM развиваться как технологически, так и методологически.

Figure 1: A comparison of BPM and SOA.

На первой конференции по Web 2.0 (first O'Reilly Media Web 2.0 conference in 2004) Тим О'Рейли сказал: Веб 2.0 это бизнес-революция в компьютерной индустрии, вызванная развитием сети Интернет как вычислительной платформы и попытка понять правила достижения успеха на этой платформе. Основное правило таково: Строить приложения, усиливающие сетевые эффекты, позволяющие функционировать тем лучше, чем больше людей их используют.

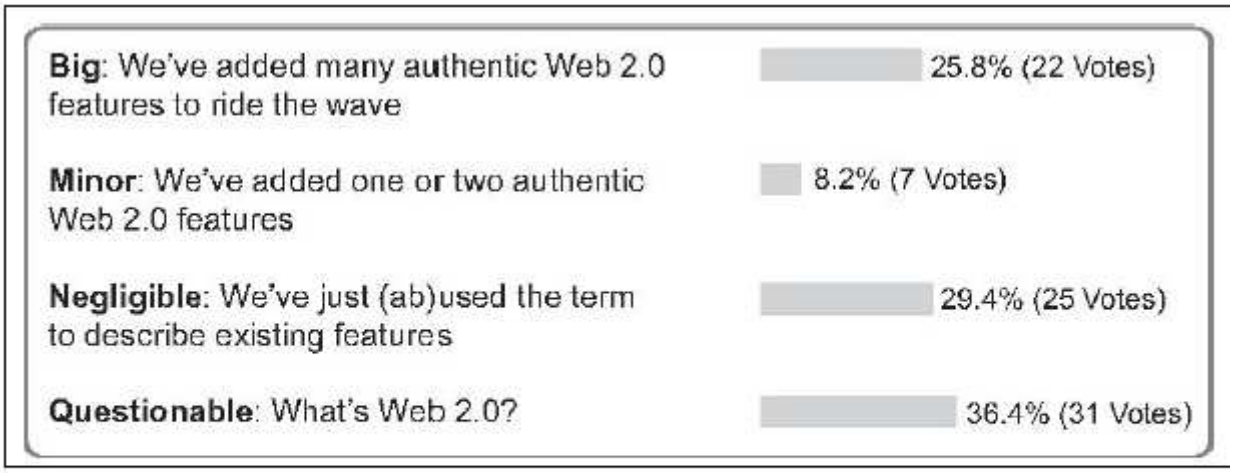


Figure 2: In a 2007 Web survey, Oracle asked its customers, What is the impact of Web 2.0 technologies on your development projects? The results shown reflect the answers of 85 respondents.

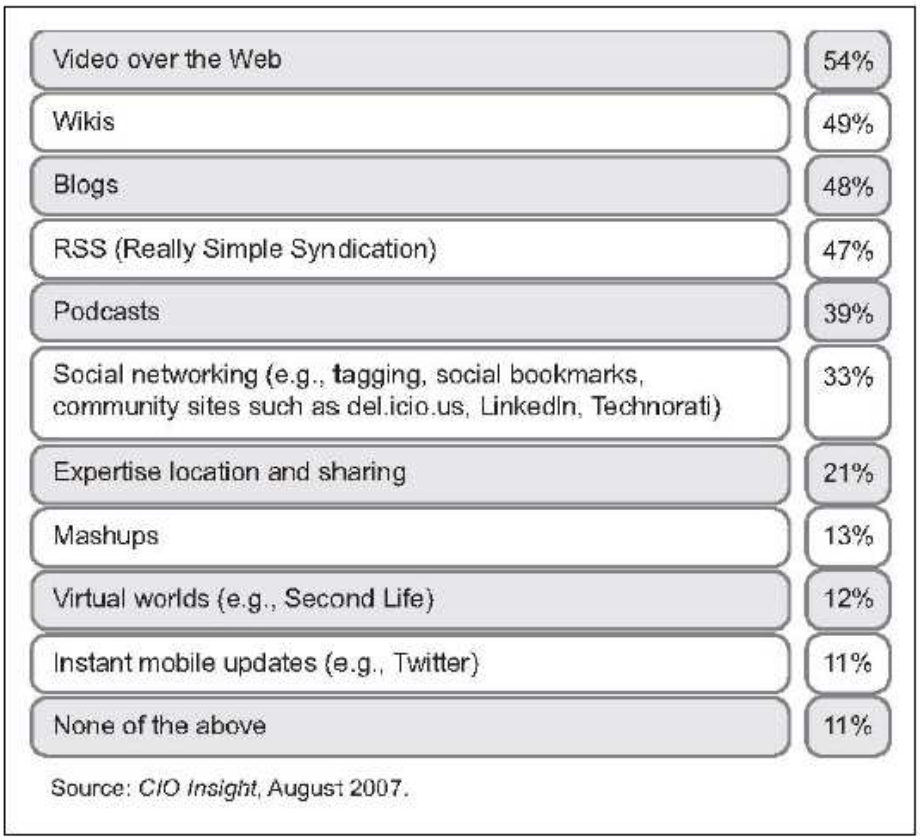


Figure 3: In 2007, *CIO Insight* asked CIOs which Web applications they used personally. The results are shown.

Схематичное представление пересечения BPM, SOA и Web2.0 требует решения множества проблем. Основная из них заключается в большом числе участников подобных проектов, их противоречивых интересов.

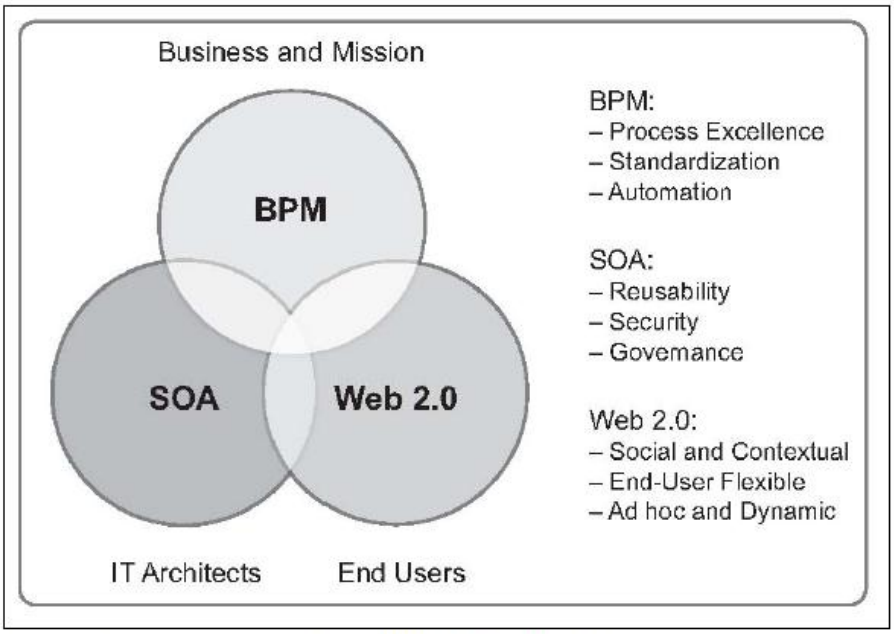


Figure 4: The points of intersection for BPM, SOA, and Web 2.0 are shown.

Управление бизнес-процессами.

Ключ к успешной трансформации бизнеса в понимании ролей всех участников. Изменения обычно требуются вследствии корректировки целей на верхнем уровне управления, но причиной может быть, например, изменение рыночных тенденций, желание владельца или стандартная схема роста.

Одной из основных задач является создание новых решений без разрушения работающих компонент, от которых зависит прибыль и выполнение работ. Сложность в сохранении ежедневной работоспособности системы без отказа от инноваций.

Обычно, в организации есть три отдельные группы: конечные пользователи, группа ИТ(enterprise information technology (EIT) group) и исполнительное руководство (chief executive officers (CXOs) group)

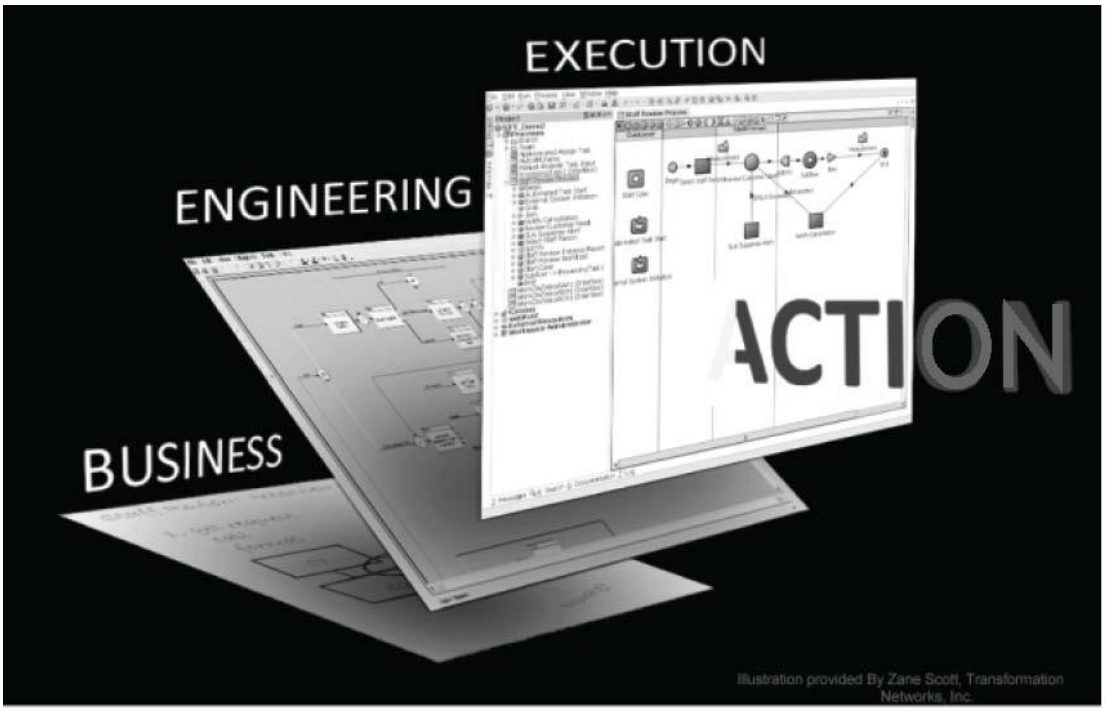
- Пользователи: используют ИС для выполнения каждодневных задач. Они не заинтересованы в модернизации технологий, с которыми работают. Они знают свои обязанности и заинтересованы тогда, когда не работает необходимое для выполнения работ, или продемонстрированы ясные преимущества новой технологии. В ряде случаев пользователи требуют сохранения старой функциональности наряду с новой.
- ИТ-специалисты, включают в себя проектировщиков, инженеров, дизайнеров. Эта группа преодолевает расхождения между руководством и конечным пользователем системы, реализуя цели бизнеса в своих решениях.
- Исполнительное руководство: эта группа занимается организацией как деятельности, так и деловой культуры предприятия, обеспечивает стратегическое планирование, достижение краткосрочных и долгосрочных целей. Эта группа может направлять культурные изменения при адаптации новой структуры предприятия.

Важнейшим фактором является наличие развитых коммуникация между участниками, с целью донести до каждого его роль в процессе.

Пользователи системы обычно видят свою роль в увеличении эффективности работы, или даже в увеличении рабочего времени, для большей отдачи. Руководство видит своей задачей адаптацию новых стратегий, правил и планов для трансформации. ИТ-специалисты занимаются обслуживанием и модернизацией информационных систем.

BPM представляет все изменения подчиненными иерархии целей, позволяя выставлять приоритеты и обеспечивая представление о создаваемой информационной системе.

Созданные модели улучшаются и обрабатываются с помощью автоматических средств визуализации. Это позволяет увидеть результат до реализации, а также определить необходимый инвентарь сервисов, который необходимо дополнительно разработать.



Большинство BPM-инструментов позволяют руководству и ИТ-специалистам работать совместно через общий инструмент и общие модели. Такие интегрированные средства разработки позволяют, например, создавать шаблоны кода, интерфейсы и примеры. Средства моделирования позволяют измерять ключевые показатели выполнения, иногда без затрат труда разработчиков, применяя некоторые средства моделирования (например, имитационное моделирование). Такие возможности позволяют менеджменту моделировать работу и отвечать на вопрос «что если..» при помощи симуляции процессов. Этим достигается совместная работа, итеративные циклы разработки, стандартизация и общий взгляд на разработку.

Тенденции: совмещение BPM, SOA и WEB 2.0

Сегодня организации осознают необходимость внедрения BPMS в областях, которые ранее считались слишком сложными для автоматизации, включая сугубо человеческие задачи с неструктурированной или плохо структурированной информацией. Не все виды деятельности могут быть формализованы, смоделированы и повторены, большая часть работ трудно автоматизировать. Люди интеллектуальных профессий часто полагаются на опыт, а не на инструкции с ограниченным числом решений и исключительных ситуаций.

Организации пытаются помочь людям интеллектуального труда применением множества разнородных систем управления знаниями и совместной работы, но такие

системы, как правило, узкоспециализированы и сложно адаптируются к конкретной задаче.

BPM позволяет обеспечивать исполнителя нужной ему информацией в требуемый момент времени, но не во всех случаях. Иногда, требуемые знания не доступны в информационной системе и их необходимо найти во внешних системах, документах и источниках данных.

Хотя основной задачей внедрения BPM являются процессы с обработкой структурированных данных и четкими, формализованными процессами, эти системы развиваются в направлении поддержки работы пользователей со знаниями, что сейчас выполняется вручную, обработкой документов, e-mail-писем и электронных таблиц.

Подобные ситуации требуют от BPM роли посредника, обеспечивающего пользователю возможность создания собственного окружения и связей с помощниками в рамках процесса.

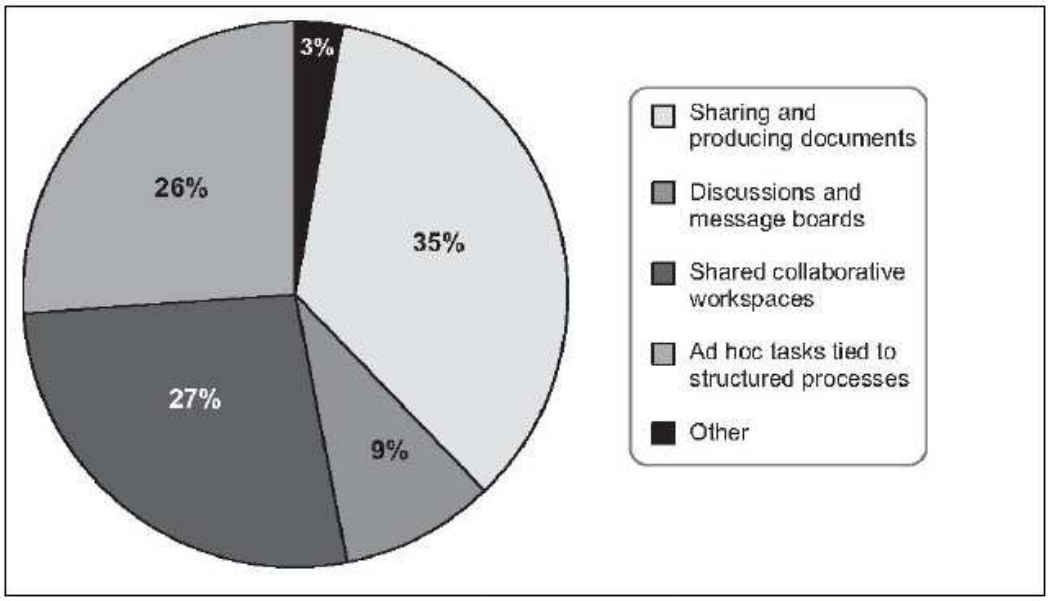


Figure 10: In a 2007 survey, Oracle customers were asked, What collaborative human activities are most important to the business process you manage or plan to manage? Their responses are shown.

Рассмотрим процесс обработки некоторой заявки. В модели процесса все выглядит простым: пользователь получает заявку, обрабатывает ее и возвращает системе результат, замыкая круг обработки. В реальности, для решения задачи, пользователю может понадобится консультация с коллегами и совместной работы.

Иногда нам заранее известно, кто эти люди, иногда их необходимо найти. Поиск людей в контексте решаемой задачи экономит время и делает труд интеллектуальных работников более эффективным.

Эффективное взаимодействие часто определяется информацией, создаваемой нами в пределах бизнеса, или во внешних источниках. Для завершения задачи необходимо иметь доступ к этим данным на этапе принятия решения.

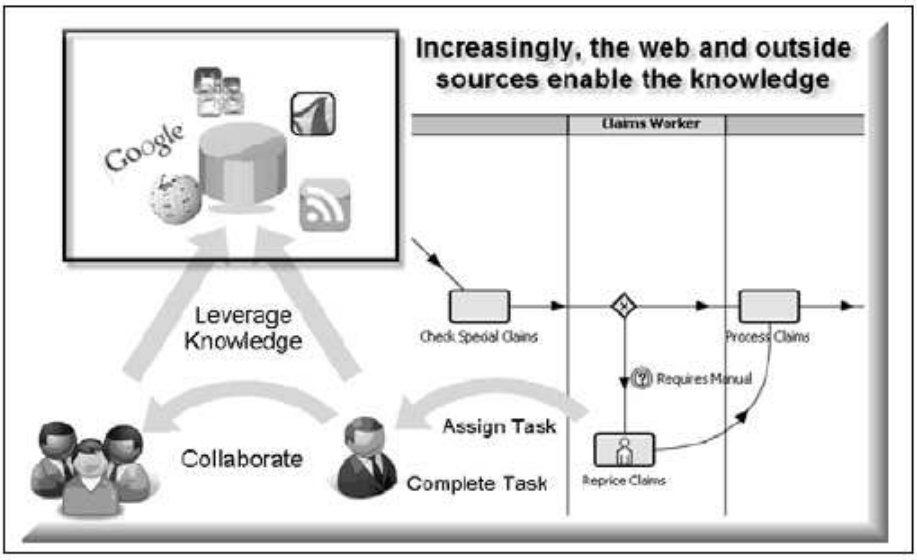


Figure 12: BPM is applied to the complex interactions and unstructured activities within knowledge worker collaborative environments.

Кроме поиска информации, люди при формировании решений используют средства и ресурсы за пределами ИС компании. Эти средства часто не совместимы с корпоративной ИС, а в худшем случае к ним нет доступа.

Для облегчения подобных взаимодействий, система моделирования бизнес-процессов должна позволять сотруднику получать доступ к корпоративным и внешним системам коллективной работы, средствам Web 2.0 и веб-сервисам для создания собственного окружения выполнения сложных задач.

Лекция 9. Моделирование бизнес-процессов.

Введение. Основные понятия

Говоря о моделировании бизнес-процессов, мы будем пользоваться терминологией сразу нескольких областей знаний, относящихся к экономике, информатике, моделированию сложных систем. Поэтому, прежде чем двигаться дальше, необходимо ввести ряд базовых понятий и определений.

Для начала попробуем разобраться, что, собственно, такое — «моделирование бизнес-процессов». ***Бизнес-процесс*** определяется как логически завершенная цепочка взаимосвязанных и повторяющихся видов деятельности, в результате которых ресурсы предприятия используются для переработки объекта (физически или виртуально) с целью достижения определенных измеримых результатов или создания продукции для удовлетворения внутренних или внешних потребителей. В качестве клиента бизнес-процесса может выступать другой бизнес-процесс. В цепочку обычно входят операции, которые выполняются по определенным ***бизнес-правилам.*** Под бизнес-правилами понимают способы реализации бизнес-функций в рамках бизнес-процесса, а также характеристики и условия выполнения бизнес-процесса.

Составляющие бизнес-процесс действия могут выполняться людьми (вручную или с применением компьютерных средств или механизмов) или быть полностью автоматизированы. Порядок выполнения действий и эффективность работы того, кто выполняет действие, определяют общую ***эффективность бизнес-процесса.*** Задачей каждого предприятия, стремящегося к совершенствованию своей деятельности, является построение таких бизнес-процессов, которые были бы эффективны и включали только действительно необходимые действия.

Термин ***моделирование*** имеет два основных значения. Во-первых, под моделированием понимают процесс построения модели как некоего представления (образа) оригинала, отражающего наиболее важные его черты и свойства. Если же модель уже построена, то моделирование — это процесс исследования (анализа) функционирования системы, вернее, ее модели. Базовой целью моделирования бизнес-процессов является описание реального хода бизнес-процессов компании. При этом необходимо определить, что является результатом выполнения процесса, кем и какие действия выполняются, каков их порядок, каково движение документов в ходе выполнения процесса, а также насколько процесс надежен (вероятность неудачного выполнения) и как он может быть расширен/модифицирован в будущем.

Обеспечить прозрачность хода бизнес-процессов важно потому, что только в этом случае ***владелец бизнес-процесса*** (сотрудник компании, управляющий ходом бизнес-процесса и несущий ответственность за его результаты и эффективность), бизнес-аналитик, руководство и другие заинтересованные стороны будут иметь ясное представление о том, как организована работа. Понимание хода существующих бизнес-процессов дает возможность судить об их эффективности и качестве и необходимо для разработки поддерживающей бизнес ИТ-инфраструктуры. Успешная разработка прикладных систем, обеспечивающих поддержку выполнения бизнес-процессов от начала до конца, возможна лишь тогда, когда сами процессы детально ясны.

*Моделью **бизнес-процесса*** называется его формализованное (графическое, табличное, текстовое, символьное) описание, отражающее реально существующую или предполагаемую деятельность предприятия. Модель, как правило, содержит следующие сведения о бизнес-процессе:

- набор составляющих процесс шагов — бизнес-функций;
- порядок выполнения бизнес-функций;

- механизмы контроля и управления в рамках бизнес-процесса;
- исполнителей каждой бизнес-функции;
- входящие документы/информацию, исходящие документы/информацию;
- ресурсы, необходимые для выполнения каждой бизнес-функции;
- документацию/условия, регламентирующие выполнение каждой бизнес-функции;
- параметры, характеризующие выполнение бизнес-функций и процесса в целом.

Для моделирования бизнес-процессов можно использовать различные методы. ***Метод,*** или ***методология,*** моделирования включает в себя последовательность действий, которые необходимо выполнить для построения модели, т. е. процедуру моделирования, и применяемую нотацию ***(язык).*** Наиболее популярной методологией бизнес-моделирования является ARIS, но также известны Catalyst компании CSC, Business Genetics, SCOR (Supply \ Chain Operations Reference), POEM (Process Oriented Enterprise Modeling) и др. Язык моделирования имеет свой ***синтаксис*** (условные обозначения различных элементов и правила их сочетания) и ***семантику*** (правила толкования моделей и их элементов).

В теории и на практике существуют различные подходы к построению и отображению моделей бизнес-процессов, основными из которых являются ***функциональный*** и ***объектно-ориентированный***. В функциональном подходе главным структурообразующим элементом является ***функция*** (бизнес-функция, действие, операция), и система представляется в виде иерархии взаимосвязанных функций. При объектно-ориентированном подходе система разбивается на набор ***объектов,*** соответствующих объектам реального мира и взаимодействующих между собой посредством посылки сообщений.

Бизнес-функция представляет собой специфический тип работы (операций, действий), выполняемой над продуктами или услугами по мере их продвижения в бизнес-процессе. Как правило, бизнес-функции определяются самой организационной структурой компании, начиная с функций высшего руководства через функции управления среднего и нижнего уровня и заканчивая функциями, возложенными на производственный персонал. Функциональный подход в моделировании бизнес-процессов сводится к построению схемы бизнес-процесса в виде последовательности бизнес-функций, с которыми связаны материальные и информационные объекты, используемые ресурсы, организационные единицы и т. п. Преимуществом функционального подхода является наглядность последовательности и логики операций в бизнес-процессах компании, а недостатком — некоторая субъективность в детализации операций.

В роли объектов при моделировании бизнес-процессов компании могут выступать конкретные предметы или реальные сущности, например клиент, заказ, услуга и т. п. Каждый объект характеризуется набором атрибутов, значения которых определяют его состояние, а также набором операций для проверки и изменения этого состояния. Объектно-ориентированный подход предполагает вначале выделение объектов, а затем определение тех действий, в которых они участвуют. При этом различают пассивные объекты (материалы, документы, оборудование), над которыми выполняются действия, и активные объекты (организационные единицы, конкретные исполнители, программное обеспечение), которые осуществляют действия. Такой подход позволяет более объективно выделить операции над объектами и решить задачу о целесообразности использования этих объектов. Недостаток объектно-ориентированного подхода состоит в меньшей наглядности конкретных бизнес-процессов.

Важным понятием любого метода моделирования бизнес-процессов являются ***связи*** (как правило, в графических нотациях их изображают в виде стрелок). Связи служат для

описания взаимоотношений объектов и/или бизнес-функций друг с другом. К числу таких взаимоотношений могут отноcиться: последовательность выполнения во времени, связь с помощью потока информации, использование другим объектом и т.д.

Модели бизнес-процессов применяются предприятиями для различных целей, что определяет тип разрабатываемой модели. ***Графическая*** модель бизнес-процесса в виде наглядной, общепонятной диаграммы может служить для обучения новых сотрудников их должностным обязанностям, согласования действий между структурными единицами компании, подбора или разработки компонентов информационной системы и т. д. Описание с помощью моделей такого типа существующих и целевых бизнес-процессов используется для оптимизации и совершенствования деятельности компании путем устранения узких мест, дублирования функций и проч. ***Имитационные*** модели бизнес-процессов позволяют оценить их эффективность и посмотреть, как будет выполняться процесс с входными данными, не встречавшимися до сих пор в реальной работе предприятия. ***Исполняемые*** модели бизнес-процессов могут быть запущены на специальном программном обеспечении для автоматизации процесса непосредственно по модели.

Поскольку модели бизнес-процессов предназначены для широкого круга пользователей (бизнес-аналитиков, рядовых сотрудников и руководства компании), а их построением часто занимаются неспециалисты в области информационных технологий, наиболее широко используются модели графического типа, в которых в соответствии с определенной методологией бизнес-процесс представляется в виде наглядного графического изображения — диаграммы, состоящей в основном из прямоугольников и стрелок. Такое представление обладает высокой, многомерной информативностью, которая выражается в различных свойствах (цвет, фон, начертание и т.д.) и атрибутах (вес, размер, стоимость, время и т.д.) каждого объекта и связи. В последние годы разработчики программных средств моделирования бизнес-процессов уделяют большое внимание преобразованию графических моделей в модели других видов, в частности в исполняемые, назначением которых является обеспечение автоматизации бизнес-процесса и интеграция работы задействованных в его исполнении информационных систем.

Согласно еще одной классификации, пришедшей из моделирования сложных систем, выделяют следующие виды моделей бизнес-процессов:

- *функциональные,* описывающие совокупность выполняемых системой функций и их входы и выходы;
- *поведенческие,* показывающие, когда и/или при каких условиях выполняются бизнес-функции, с помощью таких категорий, как состояние системы, событие, переход из одного состояния в другое, условия перехода, последовательность событий;
- *структурные,* характеризующие морфологию системы — состав подсистем, их взаимосвязи;
- *информационные,* отражающие структуры данных — их состав и взаимосвязи.

Развитие моделирования бизнес-процессов

История моделирования бизнес-процессов насчитывает уже почти столетие, хотя вплоть до начала 1990-х гг., когда термин «бизнес-процесс» вошел в широкое употребление, говорили об описании того, каким образом организация осуществляет свои функции и выполняет те или иные задачи. Развитие методов моделирования и автоматизации бизнес-процессов принято разделять на три этапа, или три «волны». Началом каждой из них явился очередной всплеск интереса к повышению эффективности деятельности предприятий и процессному управлению, происходивший каждый раз на новом качественном уровне. Основные характеристики этих этапов приведены в табл. 1.1 в сравнении с соответствующими стадиями развития информационных технологий и подходов к совершенствованию деятельности компании.

Таблица 1.1

Этапы в истории моделирования и управления бизнес-процессами

	Моделирование бизнес-процессов	Совершенствование деятельности	Информационные технологии
Первая волна	*1920–80-е гг.* Анализ способов выполнения работ Рационализация трудовых операций Модели на бумаге Низкая автоматизация	*1980-е гг.* Всеобщее управление качеством Непрерывность изменений Научный подход Последовательное совершенствование	*1970–90-е гг.* Система управления базами данных Совместное использование данных Приложения, обращающиеся к базам данных
Вторая волна	*1990-е гг.* ПО для построения диаграмм и анализа процессов в статике Ручной реинжиниринг Единовременное создание модели Автоматизация: КИС с поддержкой потоков работ (WfMS, ERP)	*1990-е гг.* Реинжиниринг бизнес-процессов Дискретность изменений Ненаучный подход Радикальное преобразование	*1990-е гг.* Распределенные вычисления Совместное использование функций Распределенные приложения
Третья волна	*2000-е гг.* Ориентированное на бизнес-процессы ПО Исполняемые модели Итеративная оптимизация Средства моделирования интегрированы в BPMS Имитационное моделирование и анализ моделей в динамике Конвертирование моделей Стандартизация методологий	*2000-е гг.* Управление бизнес-процессами (BPM) Непрерывность изменений Гибкость, адаптивность Научный подход Итеративное совершенствование	*2000-е гг.* Системы управления бизнес-процессами Совместное исполнение бизнес-процессов Распределенные бизнес-процессы

Начало первого этапа относят к 1920-м гг. XX в. и связывают с именем Фредерика Тейлора и его книгой «Принципы научного управления». В этот период впервые была осознана необходимость исследовать бизнес-процессы, описывать их в различных документах и действовать в соответствии с этими описаниями. Описание бизнес-процессов производится в текстовом, табличном и графическом виде, причем последний все более формализуется.

В период «первой волны» для моделирования бизнес-процессов используются блок-схемы, ориентированные графы, сети Петри, методологии SADT, IDEF, DFD. Блок-схемы на основе определенной в ГОСТ 19.701-90 нотации схем алгоритмов, программ, данных и систем (в англ. литературе — ANSI flowcharts) остаются и сегодня простейшим, но практически важным формальным графическим языком моделирования бизнес-процессов. Пример описания процесса с помощью блок-схемы приведен на рис. 1.1. Блок-схемы позволяют быстро и наглядно показать шаги бизнес-процесса в понятной каждому форме, однако их нотация не предусматривает формализованного описания многих деталей процесса, в частности исполнителей бизнес-функций.

О методологиях SADT и IDEF мы подробно поговорим в следующей главе. Что же касается сетей Петри, то использование этого аппарата непосредственно для описания бизнес-процессов хотя и имеет своих сторонников, но не завоевало широкой популярности, так как его графическая нотация не является интуитивно понятной (с ней сложно работать бизнес-аналитикам и менеджерам). Кроме того, есть процессы, которые невозможно описать с его помощью. Однако, забегая вперед, отметим, что сети Петри лягут в основу ряда языков, специально разработанных для моделирования бизнес-процессов в рамках «третьей волны».

В 1980-х гг. предпринимаются первые попытки автоматизации бизнес-процессов (уточним: не отдельных шагов, а хода процесса в целом) путем реализации в программном обеспечении для

управления документами — системах электронного документооборота — функций по отслеживанию последовательности выполняемых действий для автоматизации процедур утверждения и выпуска документов. Успех таких систем вдохновляет разработчиков ПО на распространение аналогичного подхода на автоматизацию других функциональных областей бизнеса.

Бизнес-моделирование выделяется в самостоятельное научно-прикладное направление только к началу 1990-х гг. Большинство созданных и применяемых до этого момента методологий не предназначались специально для описания бизнес-процессов, а разрабатывались для моделирования сложных систем и проектирования ПО. Они зачастую лишены строго определенной семантики. Модели, полученные с помощью таких методологий, как правило, воспринимаются интуитивно, и их интерпретация может меняться в зависимости от пользователя или области приложений модели. Эти модели хорошо подходят для обсуждения бизнес-процессов между сотрудниками компании и руководством, для чего они, собственно, и применялись, но не могут быть основой для работы информационной системы, так как неполны и допускают различные интерпретации.

1.1.

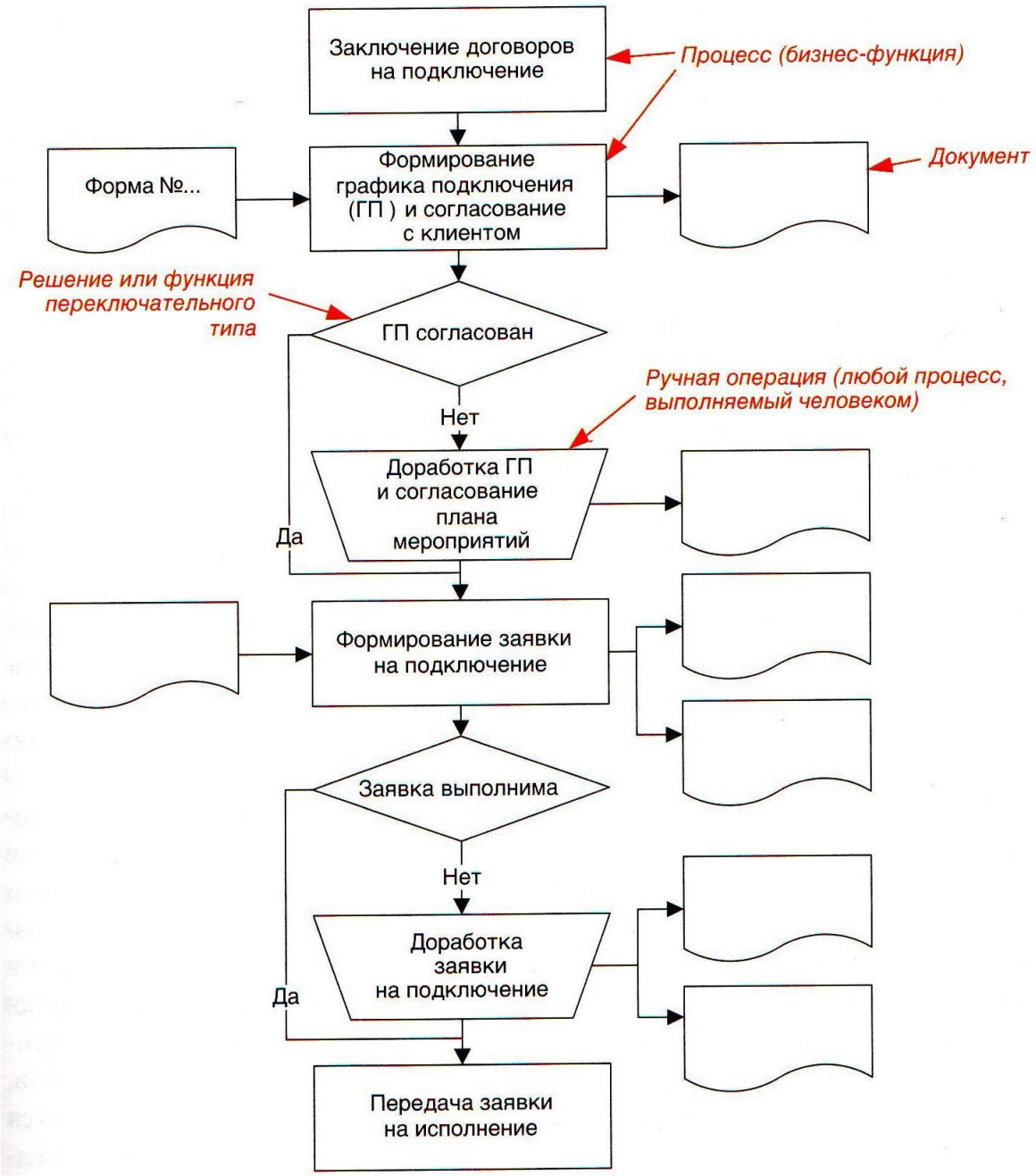


Рис. 1.1. Пример описания бизнес-процесса в виде блок-схемы

Начало второго этапа ознаменовал выход книги М. Хаммера и Д. Чампи -Реинжиниринг корпорации: манифест революции в бизнесе», которая возродила в управленческой среде интерес к описанию и анализу бизнес-процессов с целью их радикальной перестройки — *реинжиниринга.* Реинжиниринг бизнес-процессов предполагает построение двух моделей бизнес-процесса: *как есть* (англ. as is) и *как должно быть* (англ. to be), а затем внедрение последней на предприятии.

Как следующий шаг в автоматизации бизнес-процессов в 1990-х гг. появляются системы управления потоками работ WfMS (Workflow Management System) второго поколения, предназначенные для маршрутизации потоков работ любого типа в рамках бизнес-процессов компании. Эти системы снабжены средой разработчика, которая теоретически может использоваться для моделирования различных нестандартных бизнес-

процессов, однако на практике в большинстве случаев внедрение нового или изменение имеющегося процесса требовало привлечения труда программистов. Еще более ограниченные возможности по настройке и изменению процессов предоставляли поддерживающие управление потоками работ системы планирования ресурсов предприятия ERP (Enterprise Resource Planning). Внесение любых существенных изменений в бизнес-процесс превращалось в весьма дорогостоящий и долгосрочный проект по проектированию и разработке программного обеспечения, а модели бизнес-процессов, построенные аналитиками, использовались для более четкой формулировки требований, которые затем передавались программистам. В качестве примера методологии и средства автоматизации бизнес-процессов второго поколения можно назвать соответственно ARIS и распространенную ERP-систему SAP R/3.

Негибкость моделей и средств автоматизации, их неспособность обеспечить оперативное реагирование на постоянные изменения в бизнес-среде стали основными недостатками систем «второй волны», стимулировавшими разработку в начале 2000-х гг. методологий следующего — третьего — поколения. Манифестом «третьей волны» в моделировании бизнес-процессов можно по праву назвать книгу Г. Смита и П. Фингара «Управление бизнес-процессами: третья волна»[1]. На смену радикальному реинжинирингу приходит системное и «плавное» *управление.* Изменчивость бизнес-процессов, возможность их корректировки в ответ на изменения в бизнесе становятся главным критерием использования информационных технологий как средства, позволяющего получить преимущества на рынке.

Идея методологий и инструментов моделирования третьего поколения состоит в том, чтобы позволить руководству и сотрудникам компании создавать и самим внедрять новые процессы «на лету». Автоматизация процессов производится посредством так называемых *систем управления бизнес-процессами* BPMS (Business Process Management System), которые дают возможность непосредственно реализовывать бизнес-процессы в соответствии с построенной формальной моделью и не требуют разработки дополнительного программного обеспечения.

Для разработки понятных машине «исполняемых» моделей требуются более точные методы моделирования. К таким методам относятся языки моделирования на базе XML: BPML, BPEL, XPDL. Однако построение моделей непосредственно на этих языках неудобно для бизнес-пользователей. В этой связи большое внимание разработчики программного обеспечения уделяют средствам конвертирования графических моделей бизнес-процессов в исполняемые. Это позволяет бизнес-аналитику или менеджеру строить модели бизнес-процессов с использованием графической нотации, а затем преобразовывать построенную модель (пока нередко с помощью технического специалиста) в исполняемый вид.

Следует понимать, что графические модели, предназначенные для преобразования в исполняемые, должны быть гораздо более строгими и формальными по сравнению с моделями, создаваемыми в аналитических целях. Например, графическую модель, построенную в виде блок-схемы с обширными текстовыми комментариями, автоматически конвертировать в исполняемый формат не удастся. В качестве языка, позволяющего построить наглядную, понятную неподготовленному пользователю модель, которую затем можно однозначно преобразовать в исполняемый язык (изначально это был 3PML), выступила нотация BPMN. Она поддерживает описание таких < программистских» функций, как обработка событий и ошибок, откат транзакций и т. п.

«Третья волна» принесла в моделирование бизнес-процессов стремление к стандартизации. Методологии построения исполняемых моделей разрабатываются и выпускаются организациями по стандартизации и международными консорциумами:

- OASIS (Organization for the Advancement of Structured Information Standards, осн. в 1993 г.) выпускает спецификации ebXML и BPEL, а также различные стандарты для электронного бизнеса на базе XML и веб-сервисов;

- OMG (Object Management Group, осн. в 1989 г.) выпускает стандарты BPMN и UML, а также MDA и CORBA;
- W3C (World Wide Web Consortium, осн. в 1994 г.) выпускает стандарты WS-CDL, WSCI, а также спецификации XML, технологии веб-сервисов и многие другие;
- WfMC (Workflow Management Coalition, осн. в 1993 г.) выпускает стандарты Wf-XML и XPDL.

На современном этапе в круг задач моделирования и автоматизации бизнес-процессов все чаще включают автоматизацию взаимодействия предприятия с внешней средой. В модели бизнес-процесса отражают взаимодействие компании с различными внешними сущностями: клиентами, коммерческими партнерами, поставщиками, административными органами. При автоматизации процесса данные взаимодействия также стараются по возможности автоматизировать. Особенно активно развиваются технологии автоматизации межкорпоративного взаимодействия — *бизнес-бизнес* (англ. Business-to-Business, B2B).

Потребности в автоматизации бизнес-процессов взаимодействия между предприятиями возникли еще в 60-х гг. прошлого века. Первое поколение электронных систем B2B-взаимодействия описывает стандарт UN/EDIFACT, или ЭДИФАКТ ООН (Правила ООН Электронного Обмена Данными в Управлении, Торговле и на Транспорте, ISO 9735), который, несмотря на высокую конкуренцию со стороны XML-систем в последние годы, до сих пор довольно широко применяется в Европе во многих секторах экономики.

Развитие сети Интернет послужило толчком к созданию новых методов и технологий в области электронного обмена данными. Одним из наиболее удачных методов электронного обмена является появившаяся в 1998 г. методология консорциума RosettaNet. Данная технология описывает открытую платформу электронного взаимодействия, основанного на стандарте XML, и позволяет сторонам, участвующим во взаимодействии, обмениваться бизнес-информацией через Интернет. Первоначально стандарт был разработан для индустрии высоких технологий (информационные технологии и электроника), однако предложенный подход послужил основой механизмов взаимодействия предприятий и других отраслей. В рамках методологии RosettaNet разработаны стандарты более сотни процессов бизнес-взаимодействия между различными компаниями или подразделениями внутри одного предприятия. Эти стандартизованные процессы получили название процессов интерфейса взаимодействия с партнером (Partner Interface Process, PIP) и специфицируют транзакции между двумя бизнес-системами в форме диалога на основе стандарта XML.

Еще одной современной технологией автоматизации межкорпоративного взаимодействия является ebXML (Electronic Business using extensible Markup Language, ИСО 15000). Работа над технологией ebXML началась в 1999 г. по инициативе СЕФАКТ ООН (Центр ООН по поддержке процедур и практики управления, коммерции и транспорта) и консорциума OASIS, накопившего большой опыт в сфере организации ведения бизнеса в Интернете на базе XML. Целью данного проекта является разработка инфраструктуры электронного бизнеса — полного набора спецификаций, позволяющего осуществлять бизнес-взаимодействия через единообразную XML-среду. С появлением ebXML компании получили стандартизованный де-факто метод обмена данными и бизнес-сообщениями, а также единые условия информационной поддержки торговых отношений. Архитектура ebXML объединяет спецификации формата сообщений, модели бизнес-процессов, пакет синтаксически нейтральных базовых компонентов и распределенные хранилища данных (репозитории). Стандарт ebXML получает все более широкое распространение с внедрением технологии веб-сервисов (Web Services).

Основные принципы моделирования бизнес-процессов

Давайте теперь разберемся, что означает моделирование бизнес-процессов на практике. Моделирование бизнес-процессов в компании может быть направлено на решение большого числа различных задач:

- Точно определить результат бизнес-процесса и оценить его значение для бизнеса.
- Определить набор действий, составляющих бизнес-процесс. Ясное определение набора задач и действий, которые необходимо выполнить, чрезвычайно важно для детального понимания процесса.
- Определить порядок выполнения действий. Действия в рамках одного бизнес-процесса могут выполняться как последовательно, так и параллельно. Очевидно, что параллельное исполнение, если оно допустимо, позволяет сократить общее время выполнения процесса и, следовательно, повысить его эффективность.
- Произвести разделение зон ответственности: определить, а затем отслеживать, какой сотрудник или подразделение компании несет ответственность за выполнение того или иного действия или процесса в целом.
- Определить ресурсы, потребляемые бизнес-процессом. Точно зная, кто какие ресурсы использует и для каких операций, можно повысить эффективность использования ресурсов посредством планирования и оптимизации.
- Понять суть взаимодействий между участвующими в процессе сотрудниками и подразделениями компании и оценить, а затем повысить эффективность коммуникации между ними.
- Увидеть движение документов в ходе процесса. Бизнес-процессы производят и потребляют различные документы (в бумажной или электронной форме). Важно разобраться, откуда и куда идут документы или информационные потоки, и определить, оптимально ли их движение и действительно ли все они необходимы.
- Определить потенциальные узкие места и возможности для улучшения процесса, которые будут использованы позже для его оптимизации.
- Более эффективно внедрить стандарты качества, например ИСО 9000, и успешно пройти сертификацию.
- Использовать модели бизнес-процессов в качестве руководства для новых сотрудников.
- Эффективно произвести автоматизацию бизнес-процессов в целом или отдельных их шагов, включая автоматизацию взаимодействия с внешней средой — клиентами, поставщиками, партнерами.
- Разобравшись в совокупности бизнес-процессов компании, понять и описать деятельность предприятия в целом.

В свою очередь, основной задачей при моделировании бизнес-процессов компании является описание существующих в ней процессов с целью построения их моделей «как есть». Для этого необходимо собрать всю доступную информацию о процессе, которой в полной мере, как правило, владеют только сотрудники компании, непосредственно задействованные в выполнении процесса. Таким образом, мы приходим к необходимости подробного опроса *(интервьюирования)* всех задействованных в бизнес-процессе сотрудников. Следует подчеркнуть, что нельзя ограничиваться сведениями о процессе, предоставляемыми руководителем подразделения и менеджерами. Обычно только беседа с сотрудником, непосредственно осуществляющим действия в рамках описываемого бизнес-процесса, дает адекватное представление о том, как функционирует процесс в реальности.

Первый вопрос при построении модели «как есть» касается результата рассматриваемого бизнес-процесса. Случается, что получить четкую формулировку результата бизнес-процесса нелегко, несмотря на всю важность этого понятия для эффективности работы компании.

После определения результата следует разобраться с последовательностью действий, составляющих процесс. Последовательность действий моделируется на разных уровнях

абстракции. На самом верхнем уровне показывают только наиболее важные шаги процесса (обычно не более десяти). Затем производится декомпозиция каждого из высокоуровневых шагов *(подпроцессов).* Глубина декомпозиции определяется сложностью процесса и требуемой степенью детализации. Для того чтобы получить действительно полное представление о бизнес-процессе, надо произвести декомпозицию до атомарных бизнес-функций — хорошо понятных элементарных действий (отдельных операций в ПО или выполняемых человеком), которые нет смысла раскладывать на составляющие.

На основе собранной информации строится модель обычного, или оптимального, выполнения процесса и определяются возможные сценарии его выполнения со сбоями. Различные сбои (исключительные ситуации — *исключения)* могут нарушать оптимальный ход процесса, поэтому следует указать, каким образом исключения будут *«обработаны»,* то есть какие действия предпринимаются в случае возникновения исключительной ситуации.

На рис. 1.2 показаны основные шаги при построении модели бизнес-процесса.

Рис. 1.2. Шаги построения модели бизнес-процесса

Важной частью построения модели бизнес-процесса является исследование аспектов его эффективности. Сюда входят использование ресурсов, время выполнения работ сотрудниками, возможные задержки и простои. Необходимо разработать систему показателей, или метрик, для оценки эффективности процесса. Частично в качестве метрик могут быть взяты используемые в компании KPI (Key Performance Indicator), однако могут потребоваться и дополнительные характеризующие рассматриваемый процесс показатели.

При моделировании определяются *бизнес-цели,* в достижение которых вносит свой вклад моделируемый процесс. Следует различать понятия бизнес-цели и результата процесса. Каждый бизнес-процесс должен иметь как минимум один результат и быть направлен на достижение хотя бы одной бизнес-цели. Например, результат процесса «Исполнение заказа на подключение абонента» можно определить как «Получение подтверждения подключения от клиента», тогда как бизнес-цели, которые преследуются при выполнении данного процесса, могут включать «Обеспечение минимального времени исполнения заказа» и «Обеспечение минимального процента рекламаций». Для определения целей следует обратиться к бизнес-стратегии компании.

Необходимо выявить события, которые могут прервать ход процесса. В случае прерывания может потребоваться корректно «откатить» *(компенсировать)* те шаги процесса, которые уже были выполнены. Для этого следует определить логику компенсирующих действий для каждого прерывающего события.

Наконец, следует рассмотреть имеющиеся программные средства, осуществляющие поддержку бизнес-процесса. Это важно, так как программное обеспечение может скрывать некоторые особенности поведения процесса, не в полной мере известные исполняющим отдельные шаги сотрудникам. Собранная на этом этапе информация будет полезна при дальнейшей автоматизации процесса.

Собрав все указанные сведения, можно получить хорошее представление о ходе бизнес-процесса. На этапе моделирования должны быть получены следующие результаты:

- *Процессная карта,* показывающая связь между различными бизнес-процессами и их взаимодействия. На процессной карте, как правило, каждый бизнес-процесс компании изображен в виде прямоугольника, стрелками показаны связи между ними (например, зависимость одного процесса от другого, или замена одного процесса другим при выполнении некоторого условия), а также представлены различные документы, которые передаются из процесса в процесс или регламентируют их ход (стандарты, инструкции и т.п.).
- *Диаграмма ролей,* показывающая роли при выполнении процесса и связи между ними. Диаграмма ролей не является иерархической. Она представляет такие связи, как участие в группе, руководство, коммуникацию, замещение одной роли другой и т. д.
- Модель «как есть» каждого рассмотренного бизнес-процесса, детально описывающая процесс и отражающая ход процесса, действия, роли, движение документов, а также точки возможной оптимизации. Такая модель включает в себя:
  - диаграмму окружения процесса, представляющую бизнес-процесс в виде одного действия (то есть не раскрывающую ход процесса), для которого могут быть показаны запускающее процесс событие, необходимые входные данные, результат, роли, показатели эффективности, прерывающие события и компенсирующие процессы, регламентирующие документы, связанные бизнес-цели;
  - высокоуровневую диаграмму процесса, показывающую его крупные шаги (обычно не более десяти) и связанные с ними роли;
  - подробные диаграммы для каждого шага высокоуровневой модели (в зависимости от сложности процесса здесь может использоваться несколько иерархически организованных диаграмм), в деталях показывающие ход процесса, прерывающие события, бизнес-правила, роли и документы;
  - диаграмму обработки исключений, показывающую, какие действия выполняются в случае данной исключительной ситуации и кем, а также куда передается управление после окончания обработки исключения.

На практике хорошо зарекомендовал себя следующий состав группы, осуществляющей моделирование бизнес-процесса:

- владелец бизнес-процесса и один-два сотрудника того же подразделения компании, помогающих ему;
- специалист по управлению качеством;
- бизнес-аналитик(и);
- представитель ИТ-подразделения;
- внешний консультант (не обязательно).