МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Пятигорский институт (филиал) СКФУ

Методические указания к выполнению курсовой работы
по дисциплине

ТЕХНОЛОГИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Направление подготовки **09.04.02 Информационные системы и технологии**
Профиль подготовки **«Технологии работы с данными и знаниями, анализ информации»**

Квалификация выпускника **Магистр**

Пятигорск 2024 г.

Печатается по решению
Учебно-методического совета
Северо-Кавказского федерального
университета

Технологии разработки программного обеспечения: Методические указания к курсовой работе по дисциплине / сост. Флоринский О.С. – Пятигорск: Изд-во СКФУ, 2024. – 28 с.

Составитель

Флоринский О.С. – к.т.н., доцент кафедры «Систем управления и информационных технологий» ФГАОУ ВО «Северо-Кавказский федеральный университет», Пятигорский институт (филиал) СКФУ в г. Пятигорске

Методические указания содержат основные требования к написанию курсовой работы, порядок сбора и анализа материалов исследования, рекомендуемую литературу. Предназначается для студентов направления **09.04.02 «Информационные системы и технологии»**

СОДЕРЖАНИЕ

ОБЩИЕ ТРЕБОВАНИЯ К ВЫПОЛНЕНИЮ КУРСОВЫХ РАБОТ

Одной из важных форм обучения студентов является курсовое проектирование. Разработка курсовой работы по дисциплине «Технологии разработки программного обеспечения» завершает изучение этого предмета. Написание курсовой работы вырабатывает у студента навыки самостоятельной работы, навыки решения задач с использованием компьютерных систем, а также навыки системного подхода к решению задач, что является немаловажным в процессе обучения. Разработка курсовой работы является частью целенаправленного и систематизированного подхода к обучению студентов. Помогая им утвердиться в будущей специальности.

Цель курсовой работы по дисциплине «Технологии разработки программного обеспечения» – «Формирование способности проводить предпроектное обследование объекта проектирования, анализ научно-технической информации, отечественного и зарубежного опыта, анализ разработанных программных приложений, техническое проектирование и сопровождение программных приложений, разрабатывать отдельные компоненты информационной системы».

При выполнении курсовой работы студент самостоятельно осваивает все этапы создания программных приложений от постановки задачи до практической реализации.

В результате освоения данного курса студенты должны приобрести навыки по применению вычислительной техники для решения профессиональных задач.

Выполненная курсовая работа представляется на кафедру согласно графику ее выполнения. Курсовая работа регистрируется на кафедре и передается научному руководителю. В течение 10-ти дней руководитель проверяет и рецензирует работу. Если у руководителя есть замечания, то она возвращается на доработку. Студент знакомится с рецензией, вносит в случае необходимости исправления и защищает курсовую работу у руководителя.

РЕКОМЕНДАЦИИ ПО ПОДГОТОВКЕ КУРСОВОЙ РАБОТЫ

Курсовая работа выполняется на персональном компьютере и должна содержать распечатку с выполненной курсовой работой и диск, содержащий все части работы, записанные в электронном виде (каждое задание должно находиться в отдельной папке, кроме того, должен быть файл, содержащий полную электронную версию всей распечатки курсовой работы).

В распечатку курсовой работы входит: Содержание, Введение, Основная часть, состоящая из двух основных подразделов: Теоретическая часть и Практическая часть, Заключение с анализом и основными выводами, Список использованных источников, Приложения (при необходимости). Ниже представлена примерная структура содержания работы:

СОДЕРЖАНИЕ

ЗАКЛЮЧЕНИЕ
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ
ПРИЛОЖЕНИЯ (если требуются)

ВВЕДЕНИЕ (страница 3)

Во введении обосновывается актуальность выбранной темы исследования, отражаются цель и задачи, дается краткий обзор материалов теме исследования.

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Раскрытие теоретической темы курсовой работы, объем ориентировочно 20-30 страниц. Тема курсовой работы выбирается студентом индивидуально из примерной тематики теоретических заданий и согласуется с преподавателем, также студентом может быть предложена своя индивидуальная тема также согласуемая с преподавателем.

ПРАКТИЧЕСКАЯ ЧАСТЬ

Результат решения практических заданий – в выбранной студентом (и согласованной с преподавателем) среде программирования.

Вариант практического задания выбирается студентом в соответствии с последней цифрой номера зачетной книжки (студенческого билета). Студентом может быть предложена своя индивидуальная тема практического задания, которая должна быть согласована с преподавателем.

Результат решения практических заданий сдается в бумажном и электронном виде. Бумажная версия должна содержать, по каждому заданию, листинг программного кода, пояснения и распечатки всех этапов решения задач в среде программирования. Электронная версия должна содержать следующие материалы:

Задание 1 – все файлы программы и откомпилированный exe-модуль, которые должны находиться в отдельной папке «Задание 1»;
в текстовом файле (word) – листинг программного кода, пояснения и распечатки всех этапов решения задач в среде программирования;

Задание 2 – все файлы программы и откомпилированный exe-модуль, которые должны находиться в отдельной папке «Задание 2»;
в текстовом файле (word) – листинг программного кода, пояснения и распечатки всех этапов решения задач в среде программирования;

Задание 3 – все файлы программы и откомпилированный exe-модуль, которые должны находиться в отдельной папке «Задание 3»;
в текстовом файле (word) – листинг программного кода, пояснения и распечатки всех этапов решения задач в среде программирования;

Задание 4 – файл созданной базы данных, все файлы программы и откомпилированный exe-модуль, которые должны находиться в отдельной папке «Задание 4»;
в текстовом файле (word) – пояснения и распечатки всех этапов решения задач в СУБД Access. Листинг программного кода, пояснения и распечатки всех этапов решения задач в среде программирования.

Задание 5 – все файлы программы и откомпилированный exe-модуль, которые должны находиться в отдельной папке «Задание 5»;
в текстовом файле (word) - листинг программного кода, пояснения и распечатки всех этапов решения задач в среде программирования.

ЗАКЛЮЧЕНИЕ

Заключение может содержать основные выводы и анализ достигнутых результатов по теоретической и практической частям работы.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

Приводится список использованных источников (рекомендуется использовать источники не старше 5 лет).

ПРИЛОЖЕНИЯ

Приложения в работе формируется в случае их необходимости. Приложения могут включать различный дополнительный материал, необходимый для подтверждения рассматриваемых положений.

ВАРИАНТЫ ЗАДАНИЙ НА КУРСОВУЮ РАБОТУ

Теоретическая часть:

Тема курсовой работы выбирается студентом индивидуально из примерной тематики теоретических заданий и согласуется с преподавателем, также студентом может быть предложена своя индивидуальная тема также согласуемая с преподавателем.

Примерный перечень тем на курсовую работу

1. Жизненный цикл программного обеспечения.
2. Этапы проектирования программного обеспечения
3. Этап постановки задачи при разработке программного обеспечения.
4. Обеспечение диалога с пользователем при разработке программного обеспечения.
5. Отладка и тестирование разрабатываемого программного обеспечения.
6. Разработка программного обеспечения с использованием CASE – технологий.
7. Методы борьбы с ошибками в программном обеспечении
8. Проблемы обеспечения контроля качества разрабатываемого программного обеспечения.
9. Стандартизация в области разработки программного обеспечения.
10. Сертификация в области разработки программного обеспечения.

Практическая часть:

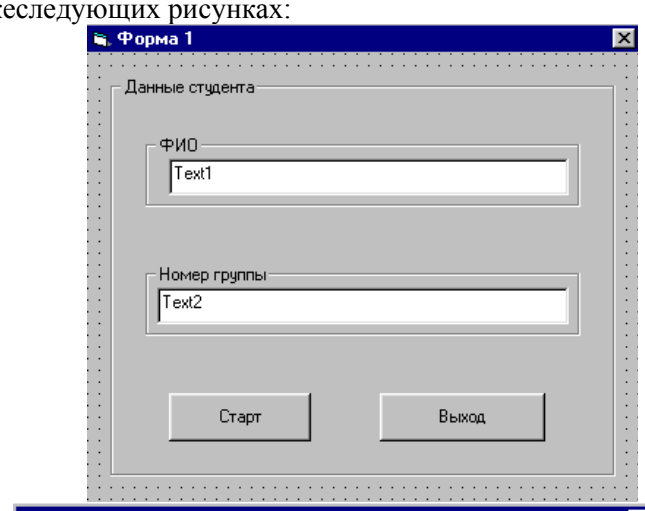
Вариант практического задания выбирается студентом в соответствии с последней цифрой номера зачетной книжки. Студентом может быть предложена своя индивидуальная тема практического задания, которая должна быть согласована с преподавателем.

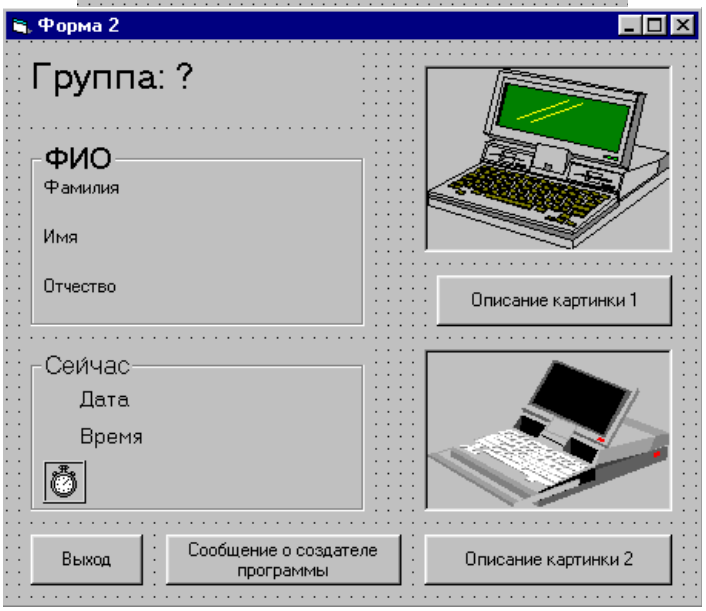
Варианты практических заданий

Вариант 1.

Задание 1.

Создайте программу, состоящую из двух форм, и внесите все необходимые элементы, как представлено на нижеследующих рисунках:





В первой форме должны заноситься ФИО и номер группы студента, при нажатии на кнопку «Старт» программа должна переходить ко второй форме.

Во второй форме:

- ФИО студента и номер группы, указанные в первой форме, должны быть отображены в соответствующих местах;
- вместо слов «Дата» и «Время» должны выводиться текущие дата и время;
- при нажатии на кнопку «Выход» программа должна закрываться;
- при нажатии на кнопку «Сообщение о создателе программы» должна появляться надпись, информирующая о создателе программы;
- при нажатии на кнопку «Описание картинки 1» должна появляться поясняющая надпись об изображении на картинке 1;
- при нажатии на кнопку «Описание картинки 2» должна появляться поясняющая надпись об изображении на картинке 2.

Задание 2.

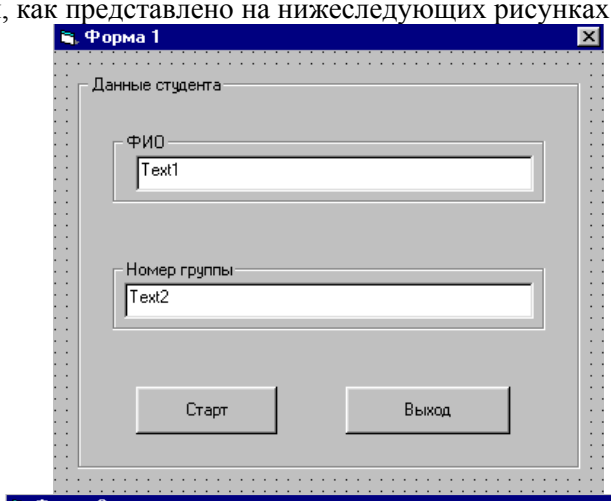
Создайте программу нахождения максимального значения Y для множества X, если известно:

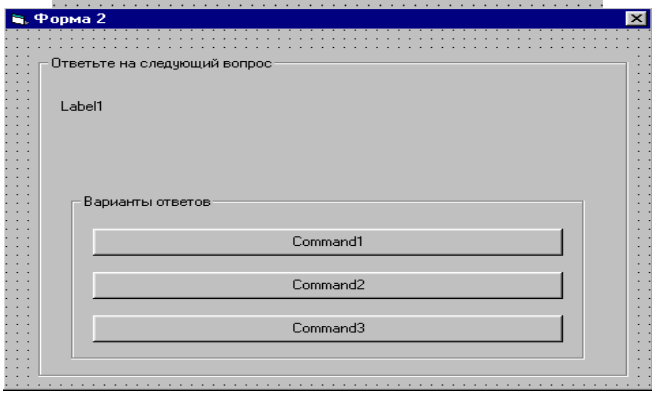
$Y=32X^2-123X+54$

Значения X начальное, X конечное и шаг должны вводиться пользователем при работе с программой. Как результат программа должна выводить максимальное значение Y и соответствующее ему значение X.

Задание 3.

Создайте программу, позволяющую тестировать студентов по дисциплине «Информатика», тема: «Текстовый процессор Word», состоящую из двух форм, и внесите все необходимые элементы, как представлено на нижеследующих рисунках:





После внесения данных в форму 1 и нажатия кнопки «Старт», должен происходить переход ко второй форме, в которой поочередно должны задаваться вопросы с тремя вариантами ответов (два неверных, один верный). После прохождения всех вопросов программа должна выдавать ФИО и номер группы тестируемого, а также результаты тестирования.

Задание 4.

Создайте с помощью MS Access базу данных по реализуемым товарам фирмы «Дежавю»; фирма реализует 20 разновидностей товара, база данных должна содержать данные, необходимые для реализации товара: наименование товара, его приходная цена, наценка, налоги, кроме того, база данных должна содержать сведения о каждом из товаров: сорт, фирма производитель, дата производства, срок хранения.

Создать с помощью средства разработки (среды программирования) программу, предоставляющую доступ к базе данных. Программа должна позволять получать доступ к данным по каждому из 20 разновидностей товара, т.е. при выборе кого-либо товара должны отображаться все данные по нему. При нажатии на кнопку «Расчет отпускной цены» программа должна выдавать следующие сведения по выбранному товару: наименование товара, приходная цена, наценка, налоги и итоговая отпускная цена.

Задание 5.

Разработайте программное обеспечение, для решения какой либо прикладной задачи. При разработке данного программного обеспечения студент должен продемонстрировать свои навыки и умения, полученные в ходе изучения литературы по технологиям программирования при освоении данной дисциплины. Выбранная тематика разрабатываемого программного обеспечения должна быть согласована с преподавателем.

Вариант 2.

Задание 1.

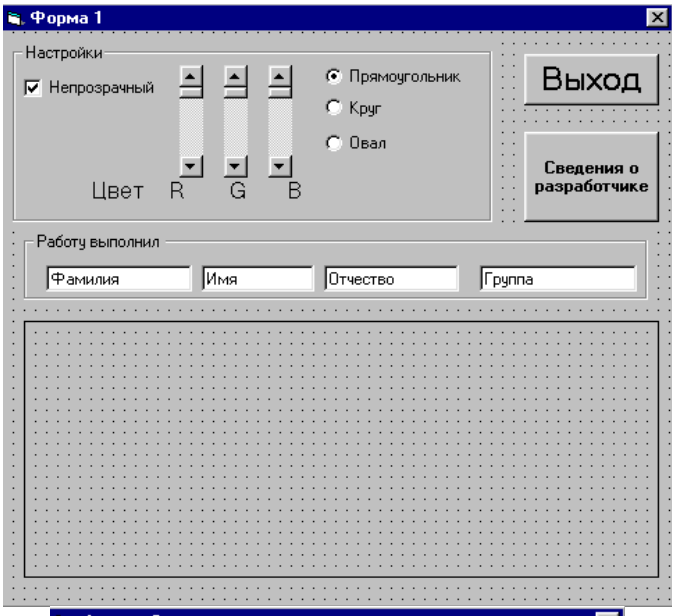
Создайте программу, позволяющую тестировать студентов по дисциплине «Информатика», тема: «Операционная система Windows». Программа должна состоять из одной формы, в которой поочередно должны задаваться вопросы (не менее 15 вопросов) с четырьмя вариантами ответов (три неверных, один верный). После прохождения всех вопросов программа должна выдавать результаты тестирования и время начала и окончания прохождения теста.

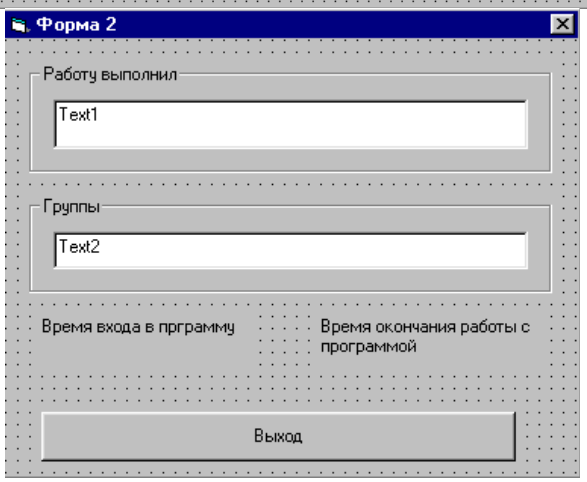
Задание 2.

Составьте программу сортировки пяти заданных чисел (2, 5, 7, 12, 8) по возрастанию.

Задание 3.

Создайте программу, состоящую из двух форм, и внесите все необходимые элементы, как представлено на нижеследующих рисунках:





В первой форме:

- должны заноситься ФИО и номер группы студента;
- В данной форме фигура, занимающая основное место должна:
  - при переключении менять форму: прямоугольник, круг и овал;
  - при установке флажка должна включаться и отключаться ее прозрачность;
  - при изменении положения полос прокрутки - менять цвет;
- При нажатии на кнопку «Сведения о разработчике» должна появляться надпись, информирующая о разработчике программы;
- при нажатии на кнопку «Выход» программа должна переходить ко второй форме.

Во второй форме:

- ФИО студента и номер группы, указанные в первой форме, должны быть отображены в соответствующих местах;
- вместо слов «Время входа в программу» должно бать отображено время открытия первой формы;
- вместо слов «Время окончания работы с программой» должно бать отображено время закрытия первой формы;
- при нажатии на кнопку «Выход» программа должна закрываться.

Задание 4.

Создайте с помощью MS Access базу данных по сотрудникам фирмы «Самоцвет»; в фирме работает 15 человек, база данных должна содержать данные, необходимые для расчета заработной платы: ФИО сотрудника, его должность, квалификационный разряд, количество отработанных дней в месяце и сумму оплаты за один день, кроме того, база данных должна содержать личные данные сотрудников: год рождения, номер паспорта, адрес проживания, телефон.

Создать с помощью средства разработки (среды программирования) программу, предоставляющую доступ к базе данных. Программа должна позволять получать доступ к данным по каждому из сотрудников фирмы, т.е. при выборе кого-либо сотрудника должны отображаться все данные по нему. При нажатии на кнопку «Расчет заработной платы» программа должна выдавать следующие сведения по выбранному сотруднику: ФИО сотрудника, общую начисленную сумму и сумму к выдаче, после вычитания налогов.

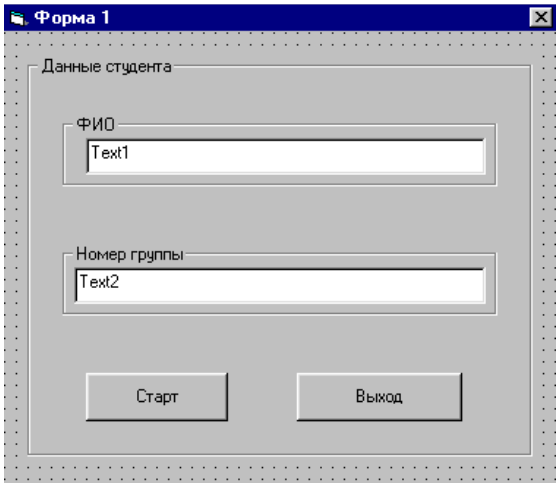
Задание 5.

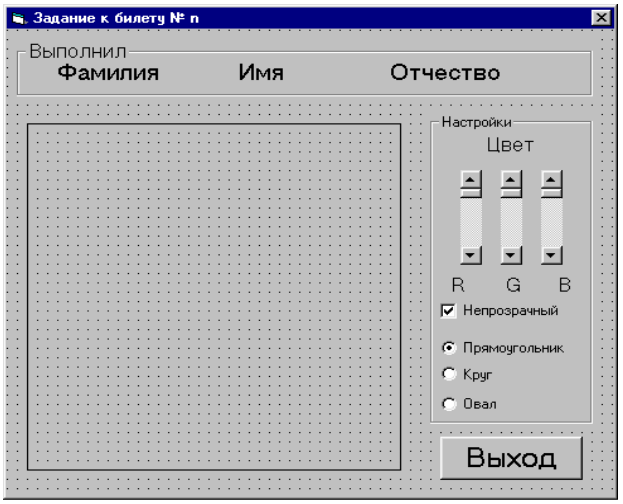
Разработайте программное обеспечение, для решения какой либо прикладной задачи. При разработке данного программного обеспечения студент должен продемонстрировать свои навыки и умения, полученные в ходе изучения литературы по технологиям программирования при освоении данной дисциплины. Выбранная тематика разрабатываемого программного обеспечения должна быть согласована с преподавателем.

Вариант 3.

Задание 1.

Создайте программу, состоящую из двух форм, и внесите все необходимые элементы, как представлено на нижеследующих рисунках:





В первой форме должны заноситься ФИО и номер группы студента, при нажатии на кнопку «Старт» программа должна переходить ко второй форме.

Во второй форме:

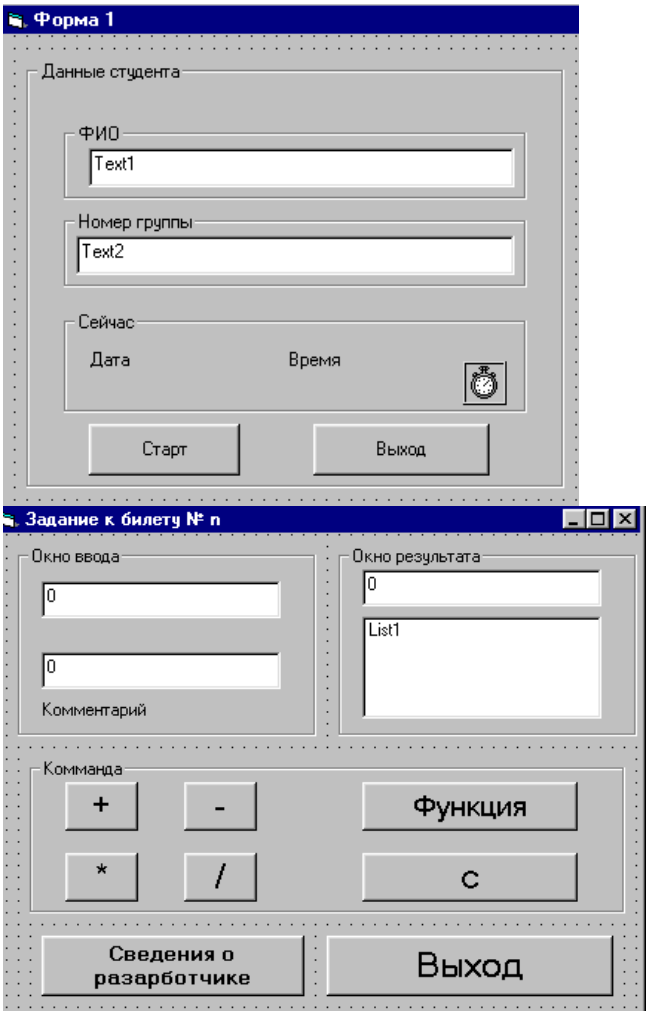
- ФИО студента, указанные в первой форме, должны быть отображены в соответствующих местах;
- при переключении менять форму: прямоугольник, круг и овал;
- при установке флажка должна включаться и отключаться ее прозрачность;
- при изменении положения полос прокрутки менять цвет;
- при нажатии на кнопку «Выход» программа перед закрытием должна выдавать время начала и окончания работы с программой.

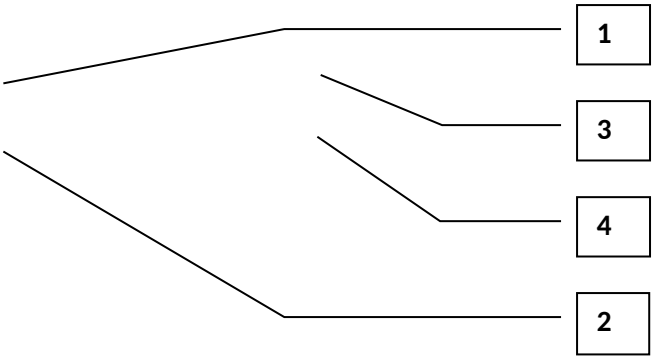
Задание 2.

Напишите программу определения наличия задаваемого слова в произвольном тексте (текст и искомое слово должны вводиться пользователем при работе с программой).

Задание 3.

Создайте программу, состоящую из двух форм, и внесите все необходимые элементы, как представлено на нижеследующих рисунках:





В первой форме:

- должны заноситься ФИО и номер группы студента;
- вместо слов «Дата» и «Время» должны выводиться текущие дата и время;
- при нажатии на кнопку «Старт» программа должна переходить ко второй форме.

Во второй форме:

- в поля «1» и «2» должны вводиться числовые значения, причем, при вводе числового значения вместо надписи «Комментарий» должна появляться надпись «ОК», а при вводе текстового значения должна появляться надпись «Введено неверное значение» и кнопки « + - * / », должны блокироваться.
- при нажатии кнопок «+» (плюс), «-» (минус), «*» (умножить), «/» (разделить) должны производиться соответствующие действия со значениями, введенными в поля «1» и «2». Результат вычислений должен отображаться в поле «3». В списке «4» должны отображаться все получаемые значения поля «3».
- при нажатии на кнопку «С» должно происходить обнуление полей «1», «2» и «3».
- при нажатии на кнопку «Функция» должна вычисляться функция по значениям полей «1» и «2», причем, в случае деления на ноль и вычислении квадратного корня из отрицательного числа программа должна выдавать сообщение об ошибке и предлагать произвести ввод данных заново:

$$A^{\frac{B}{2}} \cdot \left(\sqrt{A} - \sqrt{B}\right) \cdot \frac{\sqrt[3]{AB}}{A \cdot B}$$

- при нажатии на кнопку «Сведения о разработчике» должна появляться надпись, информирующая о разработчике программы;
- при нажатии на кнопку «Выход» программа должна закрываться.

Задание 4.

Создайте с помощью MS Access базу данных по реализуемым товарам фирмы «Апокалипсис»; фирма реализует 20 разновидностей товара, база данных должна содержать данные, необходимые для реализации товара: наименование товара, его приходная цена, наценка, налоги, кроме того, база данных должна содержать сведения о каждом из товаров: сорт, фирма производитель, дата производства, срок хранения.

Создать с помощью средства разработки (среды программирования) программу, предоставляющую доступ к базе данных. Программа должна позволять получать доступ к данным по каждому из 20 разновидностей товара, т.е. при выборе кого-либо товара должны отображаться все данные по нему. При нажатии на кнопку «Дата реализации» программа должна выдавать следующие сведения по выбранному товару: наименование товара, дата производства, срок хранения и дату, до которой должен быть реализован данный товар.

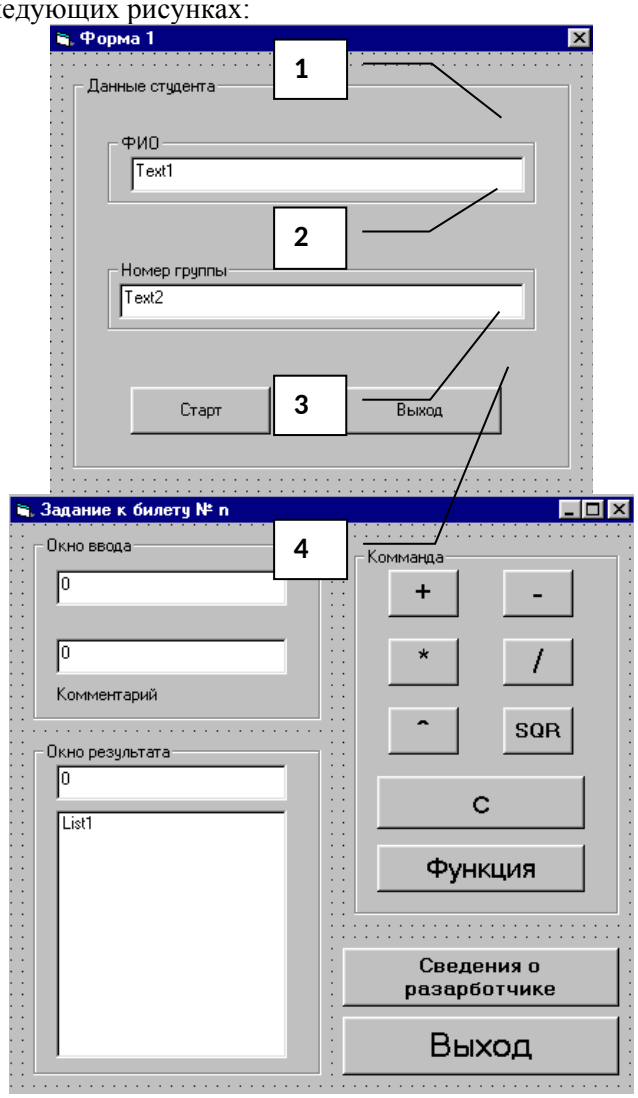
Задание 5.

Разработайте программное обеспечение, для решения какой либо прикладной задачи. При разработке данного программного обеспечения студент должен продемонстрировать свои навыки и умения, полученные в ходе изучения литературы по технологиям программирования при освоении данной дисциплины. Выбранная тематика разрабатываемого программного обеспечения должна быть согласована с преподавателем.

Вариант 4.

Задание 1.

Создайте программу, состоящую из двух форм, и внесите все необходимые элементы, как представлено на нижеследующих рисунках:



В первой форме:

- должны заноситься ФИО и номер группы студента;
- при нажатии на кнопку «Старт» программа должна переходить ко второй форме.

Во второй форме:

- в поля «1» и «2» должны вводиться числовые значения. При нажатии кнопок «+» (плюс), «-» (минус), «*» (умножить), «/» (разделить), «^» (возвести в степень), «SQR» (извлечь квадратный корень) должны производиться соответствующие действия со значениями, введенными в поля «1» и «2». Результат вычислений должен отображаться в поле «3». В списке «4» должны отображаться все получаемые значения поля «3»;
- при нажатии на кнопку «С» должно происходить обнуление полей «1, 2, 3»;
- при нажатии на кнопку «Функция» должна вычисляться нижеследующая функция по значениям полей «1» и «2», причем, в случае деления на ноль и вычислении квадратного корня из отрицательного числа программа должна выдавать сообщение об ошибке и предлагать произвести ввод данных заново:

$$\frac{A^2 \cdot \sqrt[3]{AB}}{A-B}$$

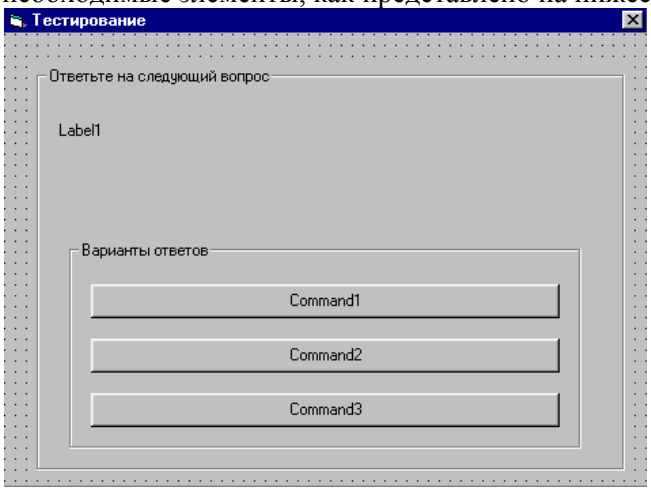
- при нажатии на кнопку «Сведения о разработчике» должна появляться надпись, информирующая о разработчике программы;
- при нажатии на кнопку «Выход» программа перед закрытием должна выдавать сообщение типа: «Студент: Иванов И. И. Группы: ИСТ – 001 завершил работу с программой», соответственно подставляя номер группы и ФИО, из указанных в первой форме.

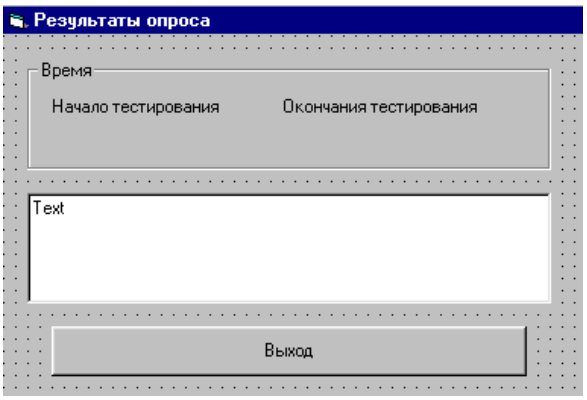
Задание 2.

Составьте программу сортировки любого количества чисел, вводимых пользователем при работе с программой.

Задание 3.

Создайте программу, позволяющую тестировать студентов по дисциплине «Информатика», тема: «Табличный процессор Excel», состоящую из двух форм, и внесите все необходимые элементы, как представлено на нижеследующих рисунках:





После запуска программы должна открываться первая форма «Тестирование», в которой поочередно должны задаваться вопросы с тремя вариантами ответов (два неверных, один верный). После прохождения всех вопросов программа должна переходить ко второй форме: «Результаты опроса», в которой вместо слов «Начало тестирования» и «Окончание тестирования» должны указываться фактическое время соответственно открытия и закрытия первой формы. Также, должны быть указаны результаты теста – количество правильных и неправильных ответов и оценка работы тестируемого. При нажатии кнопки «Выход» программа должна завершать свою работу.

Задание 4.

Создайте с помощью MS Access базу данных по студентам группы ИСТ-001; база данных должна содержать данные по успеваемости студентов: ФИО студента и его оценки по всем предметам, кроме того, база данных должна содержать личные данные студентов: год рождения, номер паспорта, адрес проживания, телефон.

Создать с помощью средства разработки (среды программирования) программу, предоставляющую доступ к базе данных. Программа должна позволять получать доступ к данным по каждому из студентов группы ИСТ-001, т.е. при выборе кого-либо студента должны отображаться все данные по нему. При нажатии на кнопку «Успеваемость» программа должна выдавать следующие сведения по выбранному студенту: ФИО студента, оценки по всем предметам и средний балл.

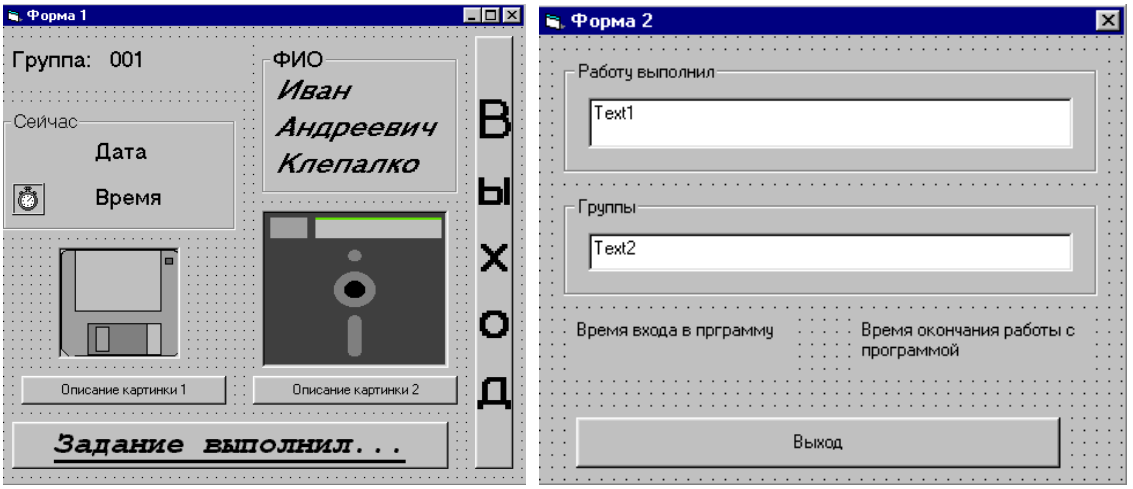
Задание 5.

Разработайте программное обеспечение, для решения какой либо прикладной задачи. При разработке данного программного обеспечения студент должен продемонстрировать свои навыки и умения, полученные в ходе изучения литературы по технологиям программирования при освоении данной дисциплины. Выбранная тематика разрабатываемого программного обеспечения должна быть согласована с преподавателем.

Вариант 5.

Задание 1.

Создайте программу, состоящую из двух форм, и внесите все необходимые элементы, как представлено на нижеследующих рисунках:



В первой форме:

- должны заноситься ФИО и номер группы студента;
- вместо слов «Дата» и «Время» должны выводиться текущие дата и время;
- при нажатии на кнопку «Задание выполнил» должна появляться надпись, о создателе программы;
- при нажатии на кнопку «Описание картинки 1» должна появляться соответствующая надпись;
- при нажатии на кнопку «Описание картинки 2» должна появляться соответствующая надпись;
- при нажатии на кнопку «Выход» программа должна переходить ко второй форме.

Во второй форме:

- ФИО студента и номер группы, указанные в первой форме, должны быть отображены в соответствующих местах;
- вместо слов «Время входа в программу» должно бать отображено время открытия первой формы;
- вместо слов «Время окончания работы с программой» должно бать отображено время закрытия первой формы;
- при нажатии на кнопку «Выход» программа должна закрываться.

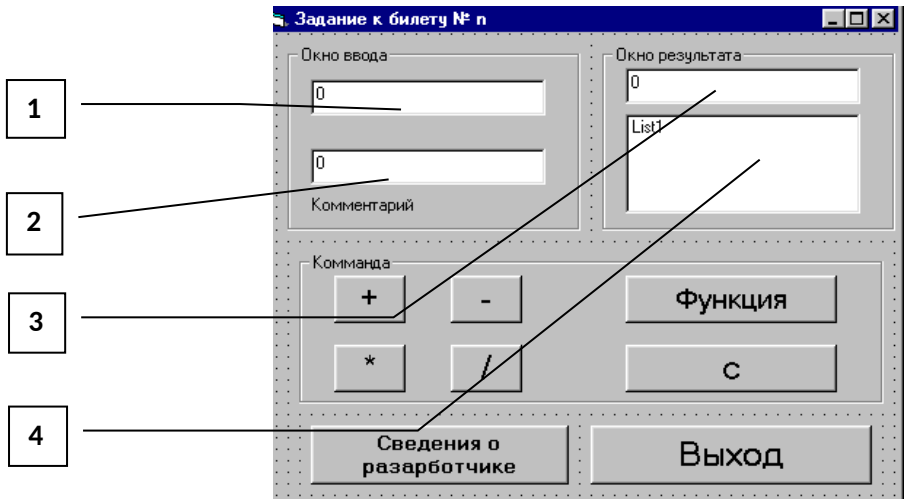
Задание 2.

Напишите программу определения наличия слова «монитор» в произвольном тексте (текст должен вводиться пользователем при работе с программой).

Задание 3.

Создайте программу, состоящую из одной формы, и внесите все необходимые элементы, как представлено на нижеследующем рисунке.

- В поля «1» и «2» должны вводиться числовые значения, причем, при вводе числового значения вместо надписи «Комментарий» должна появляться надпись «ОК», а при вводе текстового значения должна появляться надпись «Введено неверное значение» и кнопки « + - * / », должны блокироваться.
- при нажатии кнопок «+» (плюс), «-» (минус), «*» (умножить), «/» (разделить) должны производиться соответствующие действия со значениями, введенными в поля «1» и «2». Результат вычислений должен отображаться в поле «3». В списке «4» должны отображаться все получаемые значения поля «3».



- при нажатии на кнопку «Функция» должна вычисляться функция по значениям поля «1», представленная следующим выражением:

$$\text{Функция}(x) = \begin{cases} 1, & \text{при } x = 0 \\ 10, & \text{при } x = 1 \\ 50, & \text{при } x = 2 \\ \text{сообщение об ошибке, при других значениях x.} \end{cases}$$

при нажатии на кнопку «С» должно происходить обнуление полей «1, 2, 3»;

при нажатии на кнопку «Сведения о разработчике» должна появляться надпись, информирующая о разработчике программы;

дополните форму полями для ввода «Фамилии», «Имени» и «Отчества» работающего в данный момент с программой пользователя;

при нажатии на кнопку «Выход» программа перед закрытием должна выдавать сообщение типа: «Студент Иванов И. И. завершил работу с программой, время начала работы 15:25:28, время окончания работы 15:30:12», соответственно подставляя ФИО, из указанных самим пользователем в форме, а время – фактическое время начала и окончания работы с данной программой.

Задание 4.

Создайте с помощью MS Access базу данных по реализуемым товарам фирмы «Forte»; фирма реализует 20 разновидностей товара, база данных должна содержать данные, необходимые для реализации товара: наименование товара, его приходная цена, наценка, налоги, кроме того, база данных должна содержать сведения о каждом из товаров: количество товаров на складе, сорт, фирма производитель, дата производства, срок хранения.

Создать с помощью средства разработки (среды программирования) программу, предоставляющую доступ к базе данных. Программа должна позволять получать доступ к данным по каждому из 20 разновидностей товара, т.е. при выборе кого-либо товара должны отображаться все данные по нему. При нажатии на кнопку «Расчет» программа должна выдавать следующие сведения по выбранному товару: наименование товара, приходная цена всего товара данного вида, находящегося на складе, и суммарная отпускная цена этого товара на складе.

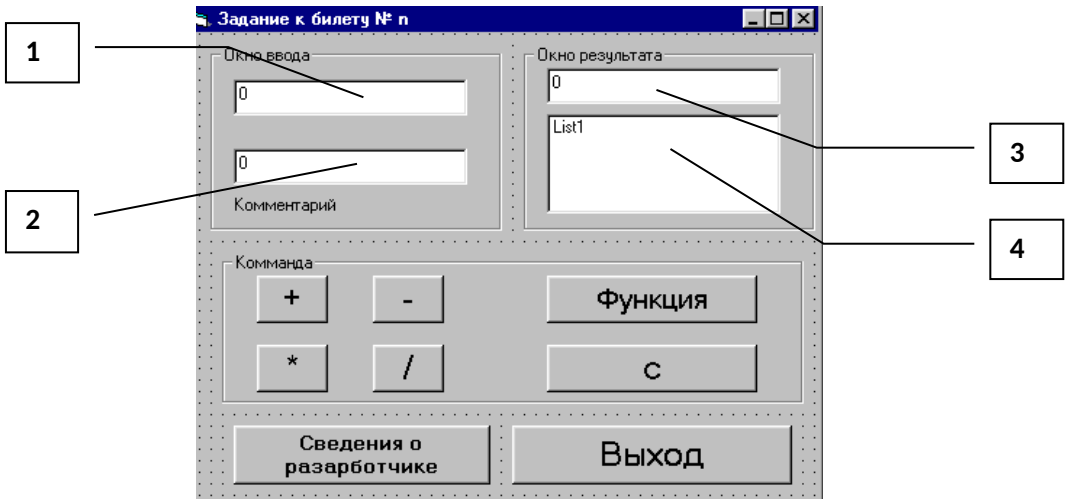
Задание 5.

Разработайте программное обеспечение, для решения какой либо прикладной задачи. При разработке данного программного обеспечения студент должен продемонстрировать свои навыки и умения, полученные в ходе изучения литературы по технологиям программирования при освоении данной дисциплины. Выбранная тематика разрабатываемого программного обеспечения должна быть согласована с преподавателем.

Вариант 6.

Задание 1.

Создайте программу, состоящую из одной формы, и внесите все необходимые элементы, как представлено на нижеследующем рисунке:



- В поля «1» и «2» должны вводиться числовые значения, причем, при вводе числового значения вместо надписи «Комментарий» должна появляться надпись «ОК», а при вводе текстового значения должна появляться надпись «Введено неверное значение» и кнопки « + - * / », должны блокироваться.
- при нажатии кнопок «+» (плюс), «-» (минус), «*» (умножить), «/» (разделить) должны производиться соответствующие действия со значениями, введенными в поля «1» и «2». Результат вычислений должен отображаться в поле «3». В списке «4» должны отображаться все получаемые значения поля «3».
- при нажатии на кнопку «Функция» должна вычисляться функция по значениям поля «1», представленная следующим выражением:

$$\text{Функция}(x) = \begin{cases} 1, & \text{при } x = 0 \\ 10, & \text{при } x = 1 \\ 50, & \text{при } x = 2 \\ \text{сообщение об ошибке, при других значениях } x. \end{cases}$$

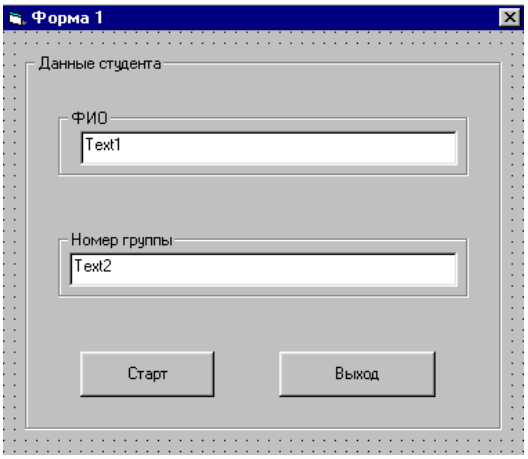
- при нажатии на кнопку «С» должно происходить обнуление полей «1, 2, 3»;
- при нажатии на кнопку «Сведения о разработчике» должна появляться надпись, информирующая о разработчике программы;
- дополните форму полями для ввода «Фамилии», «Имени» и «Отчества» работающего в данный момент с программой пользователя;
- при нажатии на кнопку «Выход» программа перед закрытием должна выдавать сообщение типа: «Студент Иванов И. И. завершил работу с программой, время начала работы 15:25:28, время окончания работы 15:30:12», соответственно подставляя ФИО, из указанных самим пользователем в форме, а время – фактическое время начала и окончания работы с данной программой.

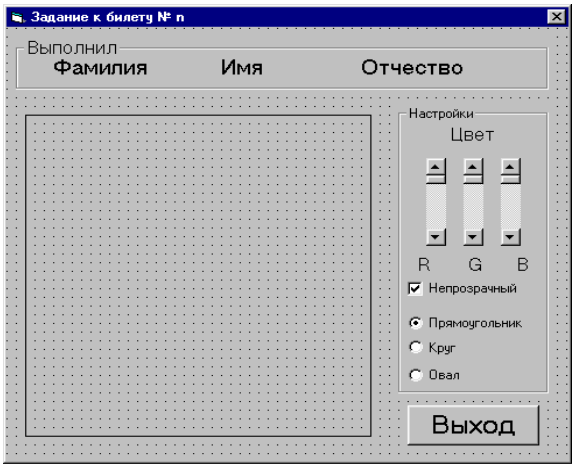
Задание 2.

Составьте программу нахождения максимального и минимального числа из любого количества чисел, вводимых пользователем при работе с программой.

Задание 3.

Создайте программу, состоящую из двух форм, и внесите все необходимые элементы, как представлено на нижеследующих рисунках:





В первой форме должны заноситься ФИО и номер группы студента, при нажатии на кнопку «Старт» программа должна переходить ко второй форме.

Во второй форме:

- ФИО студента, указанные в первой форме, должны быть отображены в соответствующих местах;
- при переключении менять форму: прямоугольник, круг и овал;
- при установке флажка должна включаться и отключаться ее прозрачность;
- при изменении положения полос прокрутки менять цвет;
- при нажатии на кнопку «Выход» программа перед закрытием должна выдавать время начала и окончания работы с программой.

Задание 4.

Создайте с помощью MS Access базу данных по реализуемым товарам фирмы «Био-Ком»; фирма реализует 20 разновидностей товара, база данных должна содержать данные, необходимые для реализации товара: наименование товара, его приходная цена, наценка, налоги, кроме того, база данных должна содержать сведения о каждом из товаров: сорт, фирма производитель, дата производства, срок хранения.

Создать с помощью средства разработки (среды программирования) программу, предоставляющую доступ к базе данных. Программа должна позволять получать доступ к данным по каждому из 20 разновидностей товара, т.е. при выборе кого-либо товара должны отображаться все данные по нему. При нажатии на кнопку «Расчет отпускной цены» программа должна выдавать следующие сведения по выбранному товару: наименование товара, приходная цена, наценка, налоги и итоговая отпускная цена.

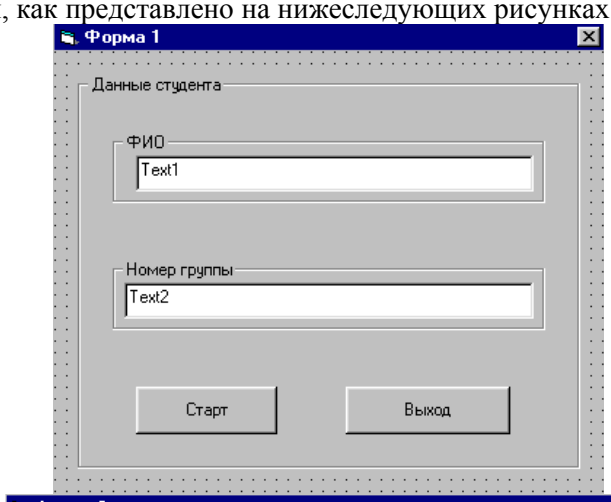
Задание 5.

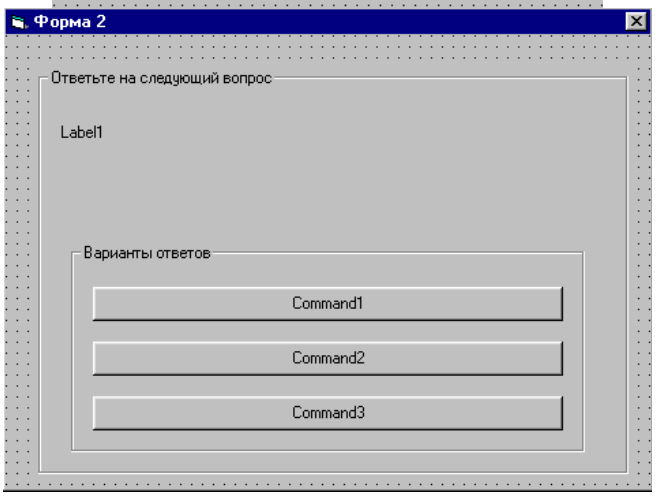
Разработайте программное обеспечение, для решения какой либо прикладной задачи. При разработке данного программного обеспечения студент должен продемонстрировать свои навыки и умения, полученные в ходе изучения литературы по технологиям программирования при освоении данной дисциплины. Выбранная тематика разрабатываемого программного обеспечения должна быть согласована с преподавателем.

Вариант 7.

Задание 1.

Создайте программу, позволяющую тестировать студентов по дисциплине «Информатика», тема: «Текстовый процессор Word», состоящую из двух форм, и внесите все необходимые элементы, как представлено на нижеследующих рисунках:





После внесения данных в форму 1 и нажатия кнопки «Старт», должен происходить переход ко второй форме, в которой поочередно должны задаваться вопросы с тремя вариантами ответов (два неверных, один верный). После прохождения всех вопросов программа должна выдавать ФИО и номер группы тестируемого, а также результаты тестирования.

Задание 2.

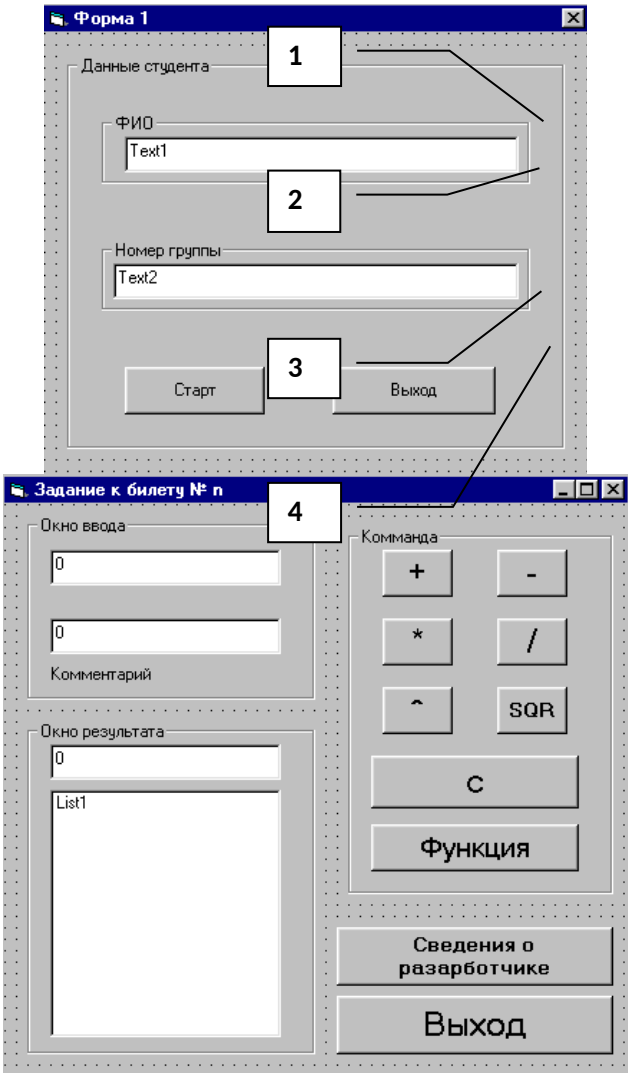
Составьте программу определения функции:

$$Y = \begin{cases} \text{Отрицательное значение, если } (X < 0); \\ \text{Критическая точка, если } (X = 0); \\ \text{Положительное значение, если } (X > 0); \end{cases}$$

Для X, изменяющегося от $X_{\text{начальное}}$ (Xn) до $X_{\text{конечное}}$ (Xk) с шагом C. Значения X начальное, X конечное и шаг должны вводиться пользователем при работе с программой.

Задание 3.

Создайте программу, состоящую из двух форм, и внесите все необходимые элементы, как представлено на нижеследующих рисунках:



В первой форме:

- должны заноситься ФИО и номер группы студента;
- при нажатии на кнопку «Старт» программа должна переходить ко второй форме.

Во второй форме:

- в поля «1» и «2» должны вводиться числовые значения. При нажатии кнопок «+» (плюс), «-» (минус), «*» (умножить), «/» (разделить), «^» (возвести в степень), «SQR» (извлечь квадратный корень) должны производиться соответствующие действия со значениями, введенными в поля «1» и «2». Результат вычислений должен отображаться в поле «3». В списке «4» должны отображаться все получаемые значения поля «3»;
- при нажатии на кнопку «С» должно происходить обнуление полей «1, 2, 3»;
- при нажатии на кнопку «Функция» должна вычисляться функция по значениям полей «1» и «2», причем, в случае деления на ноль и вычислении квадратного корня из отрицательного числа программа должна выдавать сообщение об ошибке и предлагать произвести ввод данных заново:

$$\frac{A^2 \cdot \sqrt[3]{AB}}{A-B}$$

- при нажатии на кнопку «Сведения о разработчике» должна появляться надпись, информирующая о разработчике программы;
- при нажатии на кнопку «Выход» программа перед закрытием должна выдавать сообщение типа: «Студент: Иванов И. И. Группы: ПИЭ – 001 завершил работу с программой», соответственно подставляя номер группы и ФИО, из указанных в первой форме.

Задание 4.

Создайте с помощью MS Access базу данных по сотрудникам фирмы «Кристалл»; в фирме работает 15 человек, база данных должна содержать данные, необходимые для расчета заработной платы: ФИО сотрудника, его должность, квалификационный разряд, количество отработанных дней в месяце и сумму оплаты за один день, кроме того, база данных должна содержать личные данные сотрудников: год рождения, номер паспорта, адрес проживания, телефон.

Создать с помощью средства разработки (среды программирования) программу, предоставляющую доступ к базе данных. Программа должна позволять получать доступ к данным по каждому из сотрудников фирмы, т.е. при выборе кого-либо сотрудника должны отображаться все данные по нему. При нажатии на кнопку «Расчет заработной платы» программа должна выдавать следующие сведения по выбранному сотруднику: ФИО сотрудника, общую начисленную сумму и сумму к выдаче, после вычитания налогов.

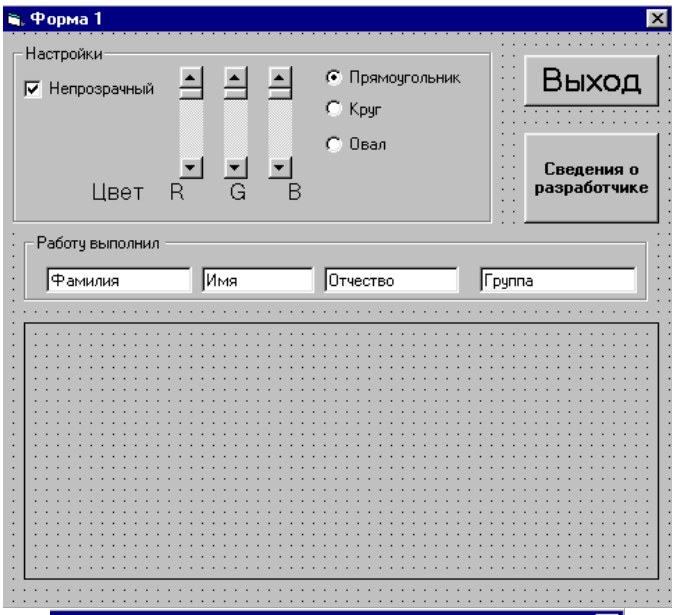
Задание 5.

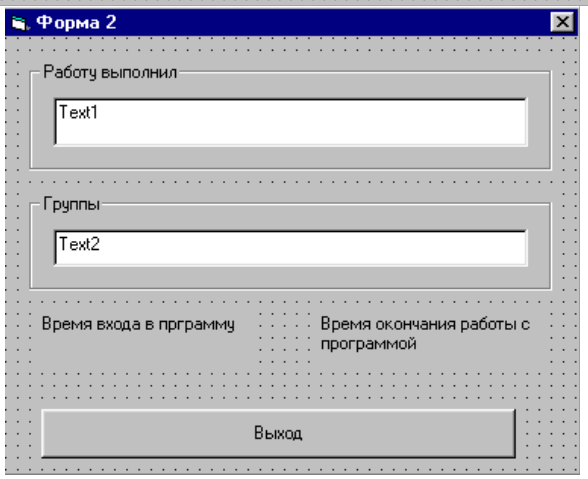
Разработайте программное обеспечение, для решения какой либо прикладной задачи. При разработке данного программного обеспечения студент должен продемонстрировать свои навыки и умения, полученные в ходе изучения литературы по технологиям программирования при освоении данной дисциплины. Выбранная тематика разрабатываемого программного обеспечения должна быть согласована с преподавателем.

Вариант 8.

Задание 1.

Создайте программу, состоящую из двух форм, и внесите все необходимые элементы, как представлено на нижеследующих рисунках:





В первой форме:

- должны заноситься ФИО и номер группы студента;
- В данной форме фигура, занимающая основное место должна:
  - при переключении менять форму: прямоугольник, круг и овал;
  - при установке флажка должна включаться и отключаться ее прозрачность;
  - при изменении положения полос прокрутки - менять цвет;
- При нажатии на кнопку «Сведения о разработчике» должна появляться надпись, информирующая о разработчике программы;
- при нажатии на кнопку «Выход» программа должна переходить ко второй форме.

Во второй форме:

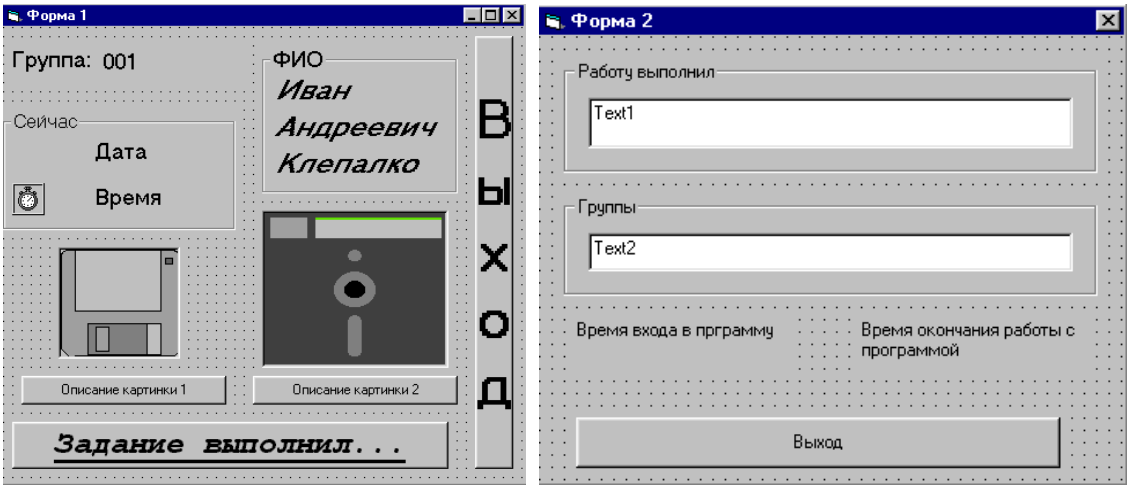
- ФИО студента и номер группы, указанные в первой форме, должны быть отображены в соответствующих местах;
- вместо слов «Время входа в программу» должно быть отображено время открытия первой формы;
- вместо слов «Время окончания работы с программой» должно быть отображено время закрытия первой формы;
- при нажатии на кнопку «Выход» программа должна закрываться.

Задание 2.

Напишите программу определения наличия слова «компьютер» в произвольном тексте (текст должен вводиться пользователем при работе с программой).

Задание 3.

Создайте программу, состоящую из двух форм, и внесите все необходимые элементы, как представлено на нижеследующих рисунках:



В первой форме:

- должны заноситься ФИО и номер группы студента;
- вместо слов «Дата» и «Время» должны выводиться текущие дата и время;
- при нажатии на кнопку «Задание выполнил» должна появляться надпись о создателе программы;
- при нажатии на кнопку «Описание картинки 1» должна появляться соответствующая надпись;
- при нажатии на кнопку «Описание картинки 2» должна появляться соответствующая надпись;
- при нажатии на кнопку «Выход» программа должна переходить ко второй форме.

Во второй форме:

- ФИО студента и номер группы, указанные в первой форме, должны быть отображены в соответствующих местах;
- вместо слов «Время входа в программу» должно быть отображено время открытия первой формы;
- вместо слов «Время окончания работы с программой» должно быть отображено время закрытия первой формы;
- при нажатии на кнопку «Выход» программа должна закрываться.

Задание 4.

Создайте с помощью MS Access базу данных по реализуемым товарам фирмы «Руно»; фирма реализует 20 разновидностей товара, база данных должна содержать данные, необходимые для реализации товара: наименование товара, его приходная цена, наценка, налоги, кроме того, база данных должна содержать сведения о каждом из товаров: сорт, фирма производитель, дата производства, срок хранения.

Создать с помощью средства разработки (среды программирования) программу, предоставляющую доступ к базе данных. Программа должна позволять получать доступ к данным по каждому из 20 разновидностей товара, т.е. при выборе кого-либо товара должны отображаться все данные по нему. При нажатии на кнопку «Дата реализации» программа должна выдавать следующие сведения по выбранному товару: наименование товара, дату производства, срок хранения и дату, до которой должен быть реализован данный товар.

Задание 5.

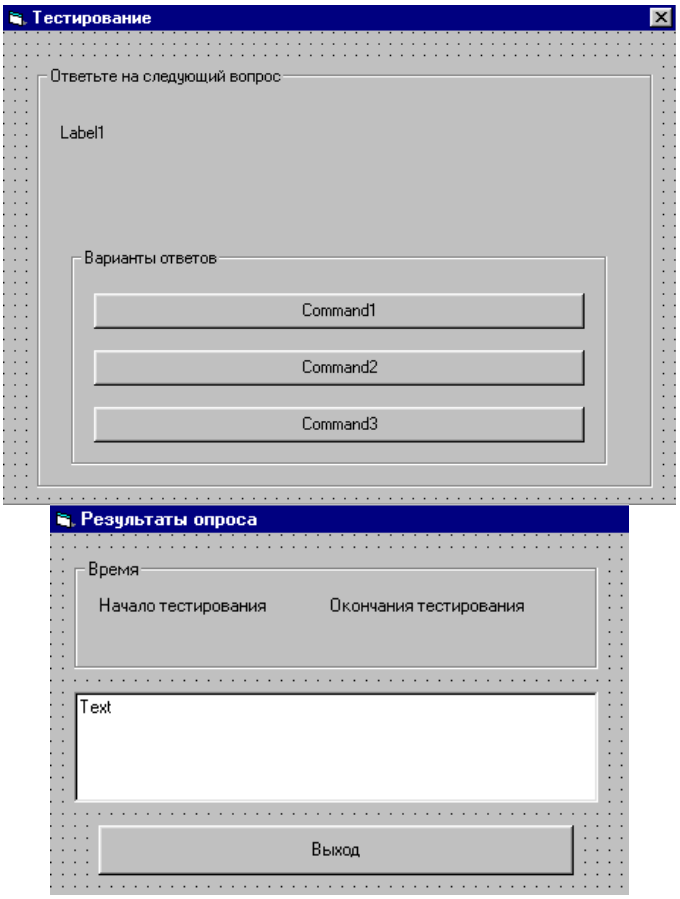
Разработайте программное обеспечение, для решения какой либо прикладной задачи. При разработке данного программного обеспечения студент должен продемонстрировать свои навыки и умения, полученные в ходе изучения литературы по технологиям программирования при

освоении данной дисциплины. Выбранная тематика разрабатываемого программного обеспечения должна быть согласована с преподавателем.

Вариант 9.

Задание 1.

Создайте программу, позволяющую тестировать студентов по дисциплине «Информатика», тема: «Табличный процессор Excel», состоящую из двух форм, и внесите все необходимые элементы, как представлено на нижеследующих рисунках:



После запуска программы должна открываться первая форма «Тестирование», в которой поочередно должны задаваться вопросы с тремя вариантами ответов (два неверных, один верный). После прохождения всех вопросов программа должна переходить ко второй форме: «Результаты опроса», в которой вместо слов «Начало тестирования» и «Окончание тестирования» должны указываться фактическое время соответственно открытия и закрытия первой формы. Также, должны быть указаны результаты теста – количество правильных и неправильных ответов и оценка работы тестируемого. При нажатии кнопки «Выход» программа должна завершать свою работу.

Задание 2.

Составьте программу нахождения значения Y, если известно:

$$15\,Y = \sum_{X=Xn}^{X=Xk} X^{\frac{1}{2X}}$$

Для X, изменяющегося от $X_{\text{начальное}}$ (Xn) до $X_{\text{конечное}}$ (Xk) с шагом C. Значения X начальное, X конечное и шаг должны вводиться пользователем при работе с программой.

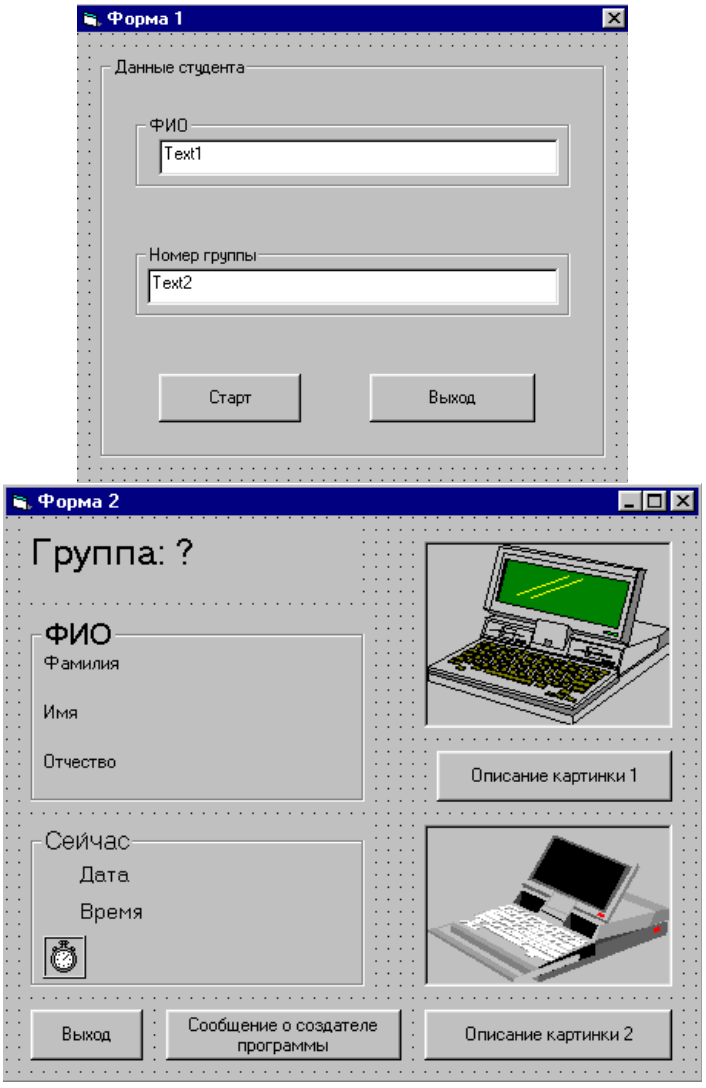
Задание 3.

Создайте программу, состоящую из двух форм, и внесите все необходимые элементы, как представлено на нижеследующих рисунках.

В первой форме должны заноситься ФИО и номер группы студента, при нажатии на кнопку «Старт» программа должна переходить ко второй форме.

Во второй форме:

- ФИО студента и номер группы, указанные в первой форме, должны быть отображены в соответствующих местах;
- вместо слов «Дата» и «Время» должны выводиться текущие дата и время;
- при нажатии на кнопку «Выход» программа должна закрываться;
- при нажатии на кнопку «Сообщение о создателе программы» должна появляться надпись, информирующая о создателе программы;
- при нажатии на кнопку «Описание картинки 1» должна появляться поясняющая надпись об изображении на картинке 1;
- при нажатии на кнопку «Описание картинки 2» должна появляться поясняющая надпись об изображении на картинке 2.



Задание 4.

Создайте с помощью MS Access базу данных по студентам группы 357 база данных должна содержать данные по успеваемости студентов: ФИО студента и его оценки по всем предметам, кроме того, база данных должна содержать личные данные студентов: год рождения, номер паспорта, адрес проживания, телефон.

Создать с помощью средства разработки (среды программирования) программу, предоставляющую доступ к базе данных. Программа должна позволять получать доступ к данным

по каждому из студентов группы 357, т.е. при выборе кого-либо студента должны отображаться все данные по нему. При нажатии на кнопку «Успеваемость» программа должна выдавать следующие сведения по выбранному студенту: ФИО студента, оценки по всем предметам и средний балл.

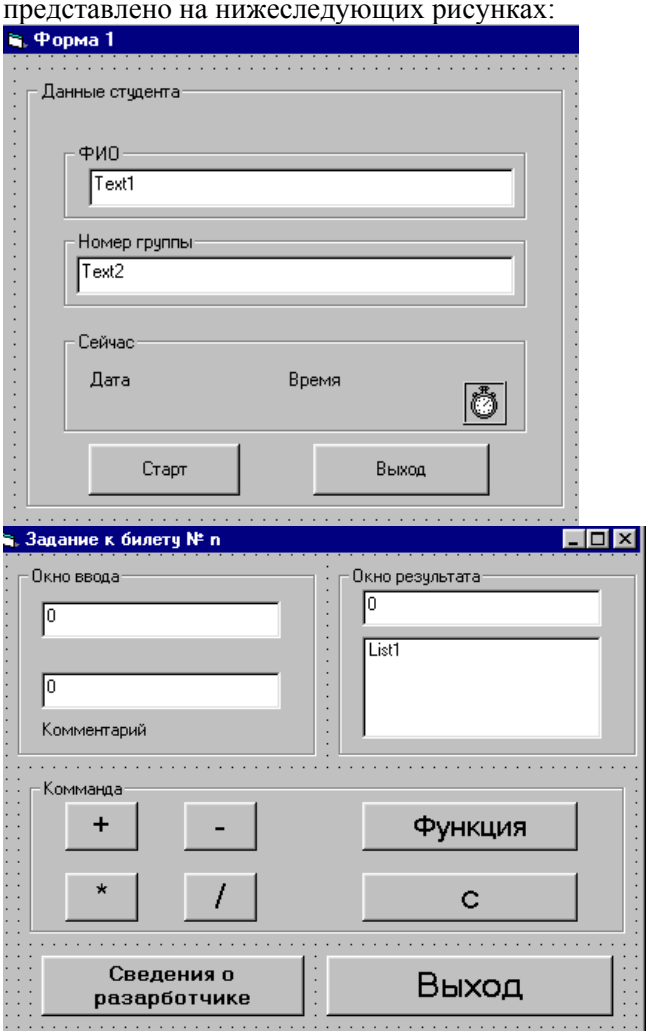
Задание 5.

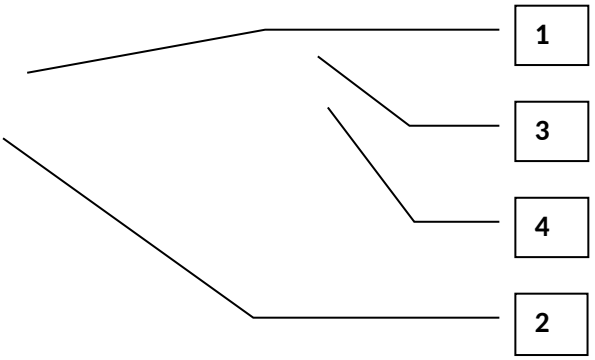
Разработайте программное обеспечение, для решения какой либо прикладной задачи. При разработке данного программного обеспечения студент должен продемонстрировать свои навыки и умения, полученные в ходе изучения литературы по технологиям программирования при освоении данной дисциплины. Выбранная тематика разрабатываемого программного обеспечения должна быть согласована с преподавателем.

Вариант 10.

Задание 1.

Создайте программу, состоящую из двух форм, и внесите все необходимые элементы, как представлено на нижеследующих рисунках:





В первой форме:

- должны заноситься ФИО и номер группы студента;
- вместо слов «Дата» и «Время» должны выводиться текущие дата и время;
- при нажатии на кнопку «Старт» программа должна переходить ко второй форме.

Во второй форме:

- в поля «1» и «2» должны вводиться числовые значения, причем, при вводе числового значения вместо надписи «Комментарий» должна появляться надпись «ОК», а при вводе текстового значения должна появляться надпись «Введено неверное значение» и кнопки « + - * / », должны блокироваться.
- при нажатии кнопок «+» (плюс), «-» (минус), «*» (умножить), «/» (разделить) должны производиться соответствующие действия со значениями, введенными в поля «1» и «2». Результат вычислений должен отображаться в поле «3». В списке «4» должны отображаться все получаемые значения поля «3».
- при нажатии на кнопку «С» должно происходить обнуление полей «1», «2» и «3».
- при нажатии на кнопку «Функция» должна вычисляться следующая функция по значениям полей «1» и «2», причем, в случае деления на ноль и вычислении квадратного корня из отрицательного числа программа должна выдавать сообщение об ошибке и предлагать произвести ввод данных заново:

$$A^{\frac{B}{2}} \cdot \left(\sqrt{A} - \sqrt{B}\right) \cdot \frac{\sqrt[3]{AB}}{A \cdot B}$$

- при нажатии на кнопку «Сведения о разработчике» должна появляться надпись, информирующая о разработчике программы;
- при нажатии на кнопку «Выход» программа должна закрываться.

Задание 2.

Составьте программу определения функции:

$$Y = \begin{cases} 15-X, \text{ если } (X < -1); \\ 15-X^2, \text{ если } (-1 <= X <= 1); \\ X^3 - 15, \text{ если } (X > 1); \end{cases}$$

Для X, изменяющегося от $X_{\text{начальное}}$ (Xn) до $X_{\text{конечное}}$ (Xk) с шагом C. Значения X начальное, X конечное и шаг должны вводиться пользователем при работе с программой.

Задание 3.

Создайте программу, позволяющую тестировать студентов по дисциплине «Информатика», тема: «СУБД MS Access». Программа должна состоять из одной формы, в которой поочередно должны задаваться вопросы (не менее 15 вопросов) с четырьмя вариантами ответов (три неверных, один верный). После прохождения всех вопросов программа должна выдавать результаты тестирования и время начала и окончания прохождения теста.

Задание 4.

Создайте с помощью MS Access базу данных по реализуемым товарам фирмы «Милена»; фирма реализует 20 разновидностей товара, база данных должна содержать данные, необходимые для реализации товара: наименование товара, его приходная цена, наценка, налоги, кроме того, база данных должна содержать сведения о каждом из товаров: количество товаров на складе, сорт, фирма производитель, дата производства, срок хранения.

Создать с помощью средства разработки (среды программирования) программу, предоставляющую доступ к базе данных. Программа должна позволять получать доступ к данным по каждому из 20 разновидностей товара, т.е. при выборе кого-либо товара должны отображаться все данные по нему. При нажатии на кнопку «Расчет» программа должна выдавать следующие сведения по выбранному товару: наименование товара, приходная цена всего товара данного вида, находящегося на складе, и суммарная отпускная цена этого товара на складе.

Задание 5.

Разработайте программное обеспечение, для решения какой либо прикладной задачи. При разработке данного программного обеспечения студент должен продемонстрировать свои навыки и умения, полученные в ходе изучения литературы по технологиям программирования при освоении данной дисциплины. Выбранная тематика разрабатываемого программного обеспечения должна быть согласована с преподавателем.

ПОСЛЕДОВАТЕЛЬНОСТЬ ВЫПОЛНЕНИЯ КУРСОВОЙ РАБОТЫ

При написании курсовой работы необходимо изучить методические рекомендации к курсовой работе, учебную литературу, сформулировав главы по своей тематике.

Предлагается придерживаться следующего примерного плана представления отдельных разделов курсовой работы (таблица 1).

Таблица 1 – Примерный план представления отдельных разделов курсовой работы

25 %	Сбор и анализ материалов необходимых для выполнения курсовой работы
50 %	Разработка теоретической части работы
75 %	Разработка практической части работы
100 %	Оформление курсовой работы

ПОРЯДОК ПРЕДСТАВЛЕНИЯ КУРСОВОЙ РАБОТЫ К ЗАЩИТЕ

Курсовая работа сдается на кафедру в установленные сроки и регистрируется в присутствии студента. Кафедра не несет ответственности за работы, сданные без регистрации. В случае пропажи таких работ они считаются несданными.

Курсовая работа проверяется преподавателем, и впоследствии допускается или не допускается к защите. Защита работы является обязательной.

В ходе защиты работы студент должен устно изложить содержание работы и ответить на дополнительные вопросы преподавателя по дисциплине.

Преподаватель дает оценку качественному уровню проделанной работы, усвоению теоретического материала.

Курсовая работа оценивается дифференцированно. Преподаватель может вернуть курсовую работу на доработку в случае неправильного её оформления или не раскрытия должным образом тематики курсовой работы. Незачтенная работа перерабатывается и сдается на повторную проверку.

Если работа выполнена не по теме закрепленной приказом кафедры, то она возвращается студенту как не выполнившему работу.

Получив проверенную курсовую работу, необходимо внимательно ознакомиться со всеми замечаниями, внести исправления. Если курсовая работа не защищена, то студент получает оценку не удовлетворительно.

Учебно-методическое и информационное обеспечение дисциплины

Перечень основной и дополнительной литературы, необходимой для освоения дисциплины (модуля)

Перечень основной литературы:

1. Битюцкая, Н. И. Разработка программных приложений : лабораторный практикум / Н. И. Битюцкая. — Ставрополь : Северо-Кавказский федеральный университет, 2015. — 140 с.

2. Абрамов, Г. В. Проектирование и разработка информационных систем : учебное пособие для СПО / Г. В. Абрамов, И. Е. Медведкова, Л. А. Коробова. — Саратов : Профобразование, 2020. — 169 с. — ISBN 978-5-4488-0730-5. — Текст : электронный // Цифровой образовательный ресурс IPR SMART : [сайт]. — URL: https://www.iprbookshop.ru/88888.html

Перечень дополнительной литературы:

1. Баженова, И. Ю. Основы проектирования приложений баз данных : учебное пособие / И. Ю. Баженова. — 3-е изд. — Москва : Интернет-Университет Информационных Технологий (ИНТУИТ), Ай Пи Ар Медиа, 2020. — 324 с. — ISBN 978-5-4497-0682-9. — Текст : электронный // Цифровой образовательный ресурс IPR SMART : [сайт]. — URL: https://www.iprbookshop.ru/97569.html

2. Гвозденко, Н. П. Разработка блок-схем алгоритмов : учебное пособие / Н. П. Гвозденко, С. А. Суслова. — Липецк : Липецкий государственный технический университет, ЭБС АСВ, 2021.

— 59 c. — ISBN 978-5-00175-055-0. — Текст : электронный // Цифровой образовательный ресурс IPR SMART : [сайт]. — URL: https://www.iprbookshop.ru/116169.html

3. Савельев, А. О. Проектирование и разработка веб-приложений на основе технологий Microsoft : учебное пособие / А. О. Савельев, А. А. Алексеев. — 4-е изд. — Москва : Интернет-Университет Информационных Технологий (ИНТУИТ), Ай Пи Ар Медиа, 2022. — 418 c. — ISBN 978-5-4497-1650-7. — Текст : электронный // Цифровой образовательный ресурс IPR SMART : [сайт]. — URL: https://www.iprbookshop.ru/120486.html

Перечень учебно-методического обеспечения самостоятельной работы обучающихся по дисциплине (модулю)

1. Методические рекомендации по выполнению лабораторных работ по дисциплине " Технология разработки программного обеспечения "

2. Методические рекомендации по организации самостоятельной работы студентов по дисциплине " Технология разработки программного обеспечения "

1. Методические указания по выполнению курсовой работы по дисциплине «Технология разработки программного обеспечения».

Перечень ресурсов информационно-телекоммуникационной сети «Интернет», необходимых для освоения дисциплины (модуля)

1. http://el.ncfu.ru/ – система управления обучением ФГАОУ ВО СКФУ. Дистанционная поддержка дисциплины «Цифровая грамотность и обработка данных»
2. http://www.un.org - Сайт ООН Информационно-коммуникационные технологии
3. http://www.intuit.ru – Интернет-Университет Компьютерных технологий.

Перечень информационных технологий, используемых при осуществлении образовательного процесса по дисциплине (модулю), включая перечень программного обеспечения и информационных справочных систем

При чтении лекций используется компьютерная техника, демонстрации презентационных мультимедийных материалов. На семинарских и практических занятиях студенты представляют презентации, подготовленные ими в часы самостоятельной работы.

Информационные справочные системы:

Информационно-справочные и информационно-правовые системы, используемые при изучении дисциплины:

1	КонсультантПлюс - http://www.consultant.ru/

Программное обеспечение:

1	Операционная система: Microsoft Windows 8: 2013-02(3000). Бессрочная лицензия. Договор № 01-эа/13 от 25.02.2013. Окончание бесплатной поддержки – 2024-01 ИЛИ Операционная система: Microsoft Windows 10: 2016-08(20), 2017-10(67), 2018-01(18), 2018-04(6), 2018-05(6), 2019-02(7). Бессрочная лицензия. Договоры № 27-эа/16 от 02.08.2016. и № 0321100021117000009_229123 от 10.10.2017. На текущий момент окончания поддержки не анонсировано.
2	Базовый пакет программ Microsoft Office (Word, Excel, PowerPoint). MicrosoftOfficeStandard 2013: договор № 01-эа/13 от 25.02.2013г., Лицензирование Microsoft Office https://support.microsoft.com/ru-ru/lifecycle/search/16674 Дата начала жизненного цикла 09.01.2013г.; набор обновлений Office 2013 Service Pack1 Дата начала жизненного цикла 25.02.2014г., Дата окончания основной фазы поддержки 10.04.2018; Дополнительная дата окончания поддержки 11.04.2024г.

Описание материально-технической базы, необходимой для осуществления образовательного процесса по дисциплине (модулю)

Лабораторные занятия	Учебная аудитория для проведения занятий семинарского типа (практических работ) — компьютерная аудитория	Количество рабочих мест – 12 Оборудование: Персональные компьютеры, объединенные в локальную сеть и имеющие выход в интернет
Самостоятельная работа	Помещения для самостоятельной работы	Компьютеры с выходом в Интернет и обеспечением доступа в электронную информационно-образовательную среду ИСУ СКФУ.

Учебные аудитории для проведения учебных занятий, оснащены оборудованием и техническими средствами обучения. Помещения для самостоятельной работы обучающихся, оснащенные компьютерной техникой с возможностью подключения к сети "Интернет" и обеспечением доступа к электронной информационно-образовательной среде. Специализированная мебель и технические средства обучения, служащие для представления учебной информации.

Материально-техническая база обеспечивает проведение всех видов дисциплинарной и междисциплинарной подготовки, лабораторной, научно-исследовательской работы обучающихся (переносной ноутбук, переносной проектор, компьютеры с необходимым программным обеспечением и выходом в интернет).

Помещения для самостоятельной работы обучающихся оснащены компьютерной техникой с возможностью подключения к сети «Интернет» и обеспечением доступа в электронную информационно-образовательную среду образовательной организации.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Пятигорский институт (филиал) СКФУ

Методические указания
по выполнению лабораторных работ
по дисциплине «**ТЕХНОЛОГИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ»**

Направление подготовки/специальность **09.04.02 Информационные системы и технологии**
Направленность (профиль)/специализация **Технологии работы с данными и знаниями, анализ информации**
Форма обучения **очная, заочная**
Год начала обучения **2024**
Изучается в 3 семестре

СОДЕРЖАНИЕ

1. ЦЕЛЬ И ЗАДАЧИ ОСВОЕНИЯ ДИСЦИПЛИНЫ

Целью изучения дисциплины «Технологии разработки программного обеспечения» учебного плана подготовки магистров по направлению подготовки 09.04.02 «Информационные системы и технологии», является получение компетенций, достаточных для анализа требований к программным системам, их документирования, проектирования, разработки, тестирования, внедрения, управления программными проектами и управления качеством разработки программных систем.

Задачами курса является приобретение и развитие знаний, умений и навыков для производственно-технологической, организационно-управленческой, проектной и научно-исследовательской деятельности.

2. **Перечень планируемых результатов обучения по дисциплине (модулю), соотнесённых с планируемыми результатами освоения образовательной программы**

Код, формулировка компетенции	Код, формулировка индикатора	Планируемые результаты обучения по дисциплине (модулю), характеризующие этапы формирования компетенций, индикаторов
ПК-2 способен создания технической документации информационно-методического и маркетингового назначения в сфере информационных технологий и систем	**ИД-1 ПК-2 С**оздает техническую документации информационно-методического назначения в сфере информационных технологий и систем **ИД-2 ПК-2** Применяет техническую документацию для создания информационных технологий и систем **ИД-3 ПК-2** Использует техническую документацию для решения маркетинговых задач в сфере информационных технологий и систем;	Проводит предпроектное обследование объекта проектирования, анализ научно-технической информации, отечественного и зарубежного опыта
ПК-4 способен выполнять разработку систем управления базами данных, операционных систем, организацию разработки системного программного обеспечения, интеграция разработанного системного программного обеспечения	**ИД-1 ПК-4** Выполняет разработку систем управления базами данных. **ИД-2 ПК-4** Проводить непосредственное руководство процессами разработки программного обеспечения; **ИД-3 ПК-4** Проводить организацию разработки системного программного обеспечения, интеграцию разработанного системного программного обеспечения.	Проводит анализ разработанных программных приложений
ПК-6 способен проводить организационное сопровождение разработки, отладки, модификации и поддержки информационных технологий и систем	**ИД-1 ПК-6** Проводить организационное сопровождение процессом разработки ПО **ИД-2 ПК-6** Выполняет управление проектами в области ИТ любого масштаба в условиях высокой неопределенности **ИД-3 ПК-6** Проводить отладку, модификацию и	Проводит техническое проектирование и сопровождение программных приложений

	поддержку информационных технологий и систем	
ПК-10 способен выполнять управление аналитическими работами и подразделением	**ИД-1 ПК-10** Выполняет разработку новых инструментов и методов управления проектами в области ИТ. **ИД-2 ПК-10** Проводить разработку новых инструментов; **ИД-3 ПК-10** Использует методы управления проектами в области ИТ	Разрабатывает отдельные компоненты информационной системы

3. НАИМЕНОВАНИЕ ЛАБОРАТОРНЫХ ЗАНЯТИЙ

№ Темы дисциплины	Наименование тем дисциплины, их краткое содержание	Очная	Заочная
2	Лабораторная работа 1. Моделирование объекта автоматизации.	4,5	**1,5**
3	Лабораторная работа 2. *Разработка модели вариантов использования и их спецификаций.*	3	**1,5**
4	Лабораторная работа 3. *Реализация архитектуры на базе объектно-реляционного отображения с типизированными объектами.*	3	**1,5**
	Итого за семестр	3	**4,5**
	Итого	**13,5**	**4,5**

4. СОДЕРЖАНИЕ ЛАБОРАТОРНЫХ ЗАНЯТИЙ

Тема 2. Анализ проблемы и постановка задачи.

Лабораторная работа № 1. Анализ проблемы, постановка задачи

Форма проведения: Практическое выполнение задания

Цель работы – сформировать навыки:

- работы с заказчиками программных систем;
- идентификации заинтересованных лиц и интервью с ними;
- анализа полученного материала;
- формулирования проблемы, ее актуальности и потребностей заинтересованных лиц.

Краткие теоретические сведения

На этапе анализа проблемы проводится анализ предметной области, для которой разрабатывается ПО. Цели этапа:

1) определение границ, или контура, системы;
2) описание объектов автоматизации и/или формализации знаний об этих объектах;
3) выявление или определение потребностей заказчика ПО.

Анализ предметной области можно проводить, например, основываясь на теории системного анализа и использовать предложенные в ней методы.

Исходными данными для этапа системного анализа являются:

1) регламенты работы отделов и должностные инструкции сотрудников этих отделов;
2) анкеты опроса заинтересованных лиц;
3) записи интервью с заинтересованными лицами;

4) другие документы, имеющие отношение к исследуемому объекту.

Выходными данными, или результатом, этапа системного анализа являются:

1) перечень заинтересованных лиц;

2) список потребностей заинтересованных лиц в разрабатываемом ПО;

3) описание объектов автоматизации;

4) модель объектов автоматизации или предметной области.

Описание примера

Здесь формулируется задача, решением которой является разработка программного обеспечения.

Итак, в результате вступления России в Болонский процесс в РФ была инициирована реформа высшего профессионального образования, в соответствии с которой Министерством образования и науки РФ была разработана программа перевода традиционной системы оценки успеваемости студентов в систему зачетных единиц (кредитов). Это объясняется необходимостью унификации систем высшего образования с целью создания единого образовательного пространства в тех странах, которые уже вступили в Болонский процесс.

В рамках этой программы все вузы страны должны к установленной дате перейти на новую систему.

Некоторый вуз, будем называть его просто «Университет» начал решать поставленную перед ним задачу поэтапно. Одной из задач перехода на новую систему в «Университете» являлась автоматизация учета текущей успеваемости и промежуточных аттестаций студентов в целях унификации этого процесса на всех кафедрах и факультетах вуза, реализации возможности автоматизированного формирования отчетов, публикации на сайте вуза рейтингов успеваемости студентов.

Составление списка заинтересованных лиц

Заинтересованные лица – это все те, кто имеет прямое или косвенное отношение к процессу, автоматизация которого производится.

Для выявления заинтересованных лиц необходимо ответить на следующие вопросы:

кто является пользователем системы?

кто является заказчиком (покупателем) системы?

на кого еще окажут влияние результаты работы системы?

кто будет оценивать и принимать систему, когда она будет представлена и развернута?

существуют ли другие внутренние или внешние пользователи системы, чьи потребности необходимо учесть?

кто будет заниматься сопровождением новой системы?

не забыли ли мы кого-нибудь?

В нашем примере определим будущих пользователей системы – это преподаватели, секретари кафедр и деканатов, заведующие кафедрами, системный администратор и сотрудники Учебного управления. Заказчиком нашей системы является вуз в лице первого проректора.

Теперь попытаемся выяснить, на кого еще будут оказывать влияние результаты работы нашей системы. Во-первых, на студентов, ведь рейтинги успеваемости всех студентов будут доступны на сайте вуза. Во-вторых, по этой же причине, на родителей. В-третьих, на деканов факультетов и заместителей деканов по учебной работе, поскольку любые изменения в учебном процессе касаются их профессиональной деятельности.

Итак, рассмотрев первые три вопроса, мы практически охватили всех

заинтересованных лиц. Поскольку сопровождать систему будет разработчик этой системы, то в список заинтересованных лиц мы его не включаем. Таким образом, получаем следующий список заинтересованных лиц для нашей системы:

преподаватели;

секретари кафедр;

секретари деканатов;

заведующие кафедрами;

Учебное управление (в лице начальника);

первый проректор;

системный администратор (тот, кто будет администрировать нашу систему);

студенты;

родители студентов;

заместители деканов по учебной работе;

деканы факультетов.

Анкетирование и проведение интервью

Для выявления потребностей заказчика и описания объектов автоматизации можно проводить как анкетирование, так и интервью. Но наибольший эффект возможен только при проведении и того и другого.

Примеры анкеты и перечня вопросов для интервью приведены ниже.

Анкета для опроса заинтересованных лиц

1. Имя.
2. Наименование организации.
3. Наименование структурного подразделения.
4. Должность.
5. Кому Вы непосредственно подчиняетесь?
6. Каковы Ваши основные обязанности?
7. Что Вы в основном производите?
8. Для кого?
9. Какие документы или какую информацию можно считать входящими, или необходимыми, для Вашей деятельности?
10. Какие документы или какую информацию можно считать исходящими, или результатом Вашей деятельности?
11. Как измеряется успех Вашей деятельности?
12. Какие проблемы влияют на успешность Вашей деятельности?
13. Какие тенденции, если такие существуют, делают Вашу работу проще или сложнее?
14. Какой интерес или какие потребности у Вас есть относительно будущего решения (разрабатываемого ПО)?

Перечень вопросов для интервью

Оценка проблемы

Для каких проблем (прикладного типа) Вы ощущаете нехватку хороших решений? Назовите их. (Не забывайте спрашивать: «А еще?»)

По каждой проблеме выясняйте следующее:

почему существует эта проблема?

как она решается в настоящее время?

как заказчик (пользователь) хотел бы ее решать?

Понимание пользовательской среды

Каковы Ваши навыки в компьютерной области?

С какими типами приложений Вы имеете опыт работы?

Какая платформа используется?

Каковы Ваши планы относительно будущих платформ?

Используется ли ПО, которое имеет отношение к данной проблеме?

(Если да, то пусть о нем немного расскажут).

Каковы Ваши ожидания относительно практичности продукта?

В каком виде должна быть представлена справочная информация для пользователя (в интерактивном или печатном)?

Резюме (перечисляются основные пункты, чтобы проверить, все ли правильно вы поняли):

Итак, Вы сказали мне… (перечислите описанные заказчиком проблемы своими словами).

Адекватно ли этот список представляет проблемы, которые имеются при существующем решении?

Какие еще проблемы Вы испытываете?

Заключение аналитика. После интервью, пока его данные еще свежи в вашей памяти, зафиксируйте не менее трех потребностей или проблем с наивысшими приоритетами, выявленных вами в беседе с данным заказчиком (пользователем).

После проведения анкетирования и интервьюирования необходимо обработать собранную информацию. На основе этих данных нужно сформулировать перечень потребностей заказчиков, построить модель предметной области и описать объект/объекты автоматизации. Все эти результаты в дальнейшем будут использованы при написании технического задания (ТЗ) на разрабатываемую систему.

Допустим, что в качестве примера было проведено анкетирование и интервью с начальником Учебного управления. Для этого, мы воспользовались одним перечнем вопросов,

совместив две эти процедуры воедино. По сути, мы провели только интервью. Предполагаемый пример результата интервью (вопросы и ответы) приведен в таблице 1.1.

Таблица 1.1 - Интервью с начальником учебного отдела

Вопрос	**Ответ**
1. Имя	Иванов Иван Иванович
2. Наименование структурного подразделения	Учебное управление
3. Должность	Начальник Учебного управления
4. Кому Вы непосредственно подчиняетесь?	Первому проректору
5. Каковы Ваши основные обязанности?	Обеспечение общего управления учебной деятельностью. Обеспечение качества и своевременного выполнения возложенных на Учебное управление задач и функций. Обеспечение выполнения мероприятий по защите конфиденциальной и информационной безопасности в подразделениях управления.
6. Что Вы в основном производите?	Функции, выполняемые Учебным управлением: 1. Создание единого нормативно-справочного поля учебного процесса. 2. Организационное сопровождение учебного процесса. 3. Планирование ресурсов учебного процесса (объемов работ кафедр, площадей, численности контингента, нагрузки на условиях почасовой оплаты, бланочной документации и др.). 4. Контроль за ходом организации и анализ показателей учебного процесса (как внешних, так и внутренних). 5. Отчетность на различные уровни управления. 6. Анализ выполнения учебной нагрузки, трудоемкости отдельных параметров учебного процесса, результатов сессии, результатов государственных аттестационных комиссий. Разработка графика учебного процесса. 7. Составление расписания учебных занятий, экзаменов, зачетов и графика защит. 8. Контроль занятости аудиторного фонда, составления семестровых планов, проведения занятий, заседаний государственных аттестационных комиссий, хода сессий, делопроизводства факультетов, оформления приложений к дипломам. 9. Учет движения студентов по всем формам обучения (очное, очно-заочное и заочное, второе высшее образование, обучение по сокращенным программам).
7. Какие документы или какую информацию можно считать входящими, или необходимыми, для Вашей деятельности?	1. Приказы и инструктивные письма Минобразования России по учебно-методическим вопросам. 2. Положение об Учебном управлении. 3. Должностные инструкции сотрудников. 4. Приказы ректора по контингенту студентов (первые экземпляры). 5. Годовые планы приема абитуриентов. 6. Планы работы факультетов. 7. Отчеты председателей государственных аттестационных комиссий по всем специальностям и направлениям.
8. Какие документы или какую информацию можно считать исходящими, или результатом Вашей	1. Семестровые планы занятий. 2. Расписание учебных занятий. 3. Расписание экзаменов. 4. Отчеты вуза по учебно-методической работе за учебный год. 5. Сводные статистические отчеты вуза о движении контингента

деятельности?	студентов на начало и конец учебного года.
9. Как измеряется успех Вашей деятельности?	В настоящее время отсутствуют количественные показатели оценки деятельности управления
10. Какие проблемы влияют на успешность Вашей деятельности?	Проблемы, связанные с организацией учебного процесса. В настоящее время основной проблемой является перевод учебного процесса в систему зачетных единиц.
11. Какой интерес или какие потребности у Вас есть относительно будущего решения (разрабатываемого ПО)?	Разрабатываемая система должна быть максимально эргономичной, работать стабильно (без сбоев); отклик системы не должен вызывать у пользователей раздражения; реализуемая функциональность должна полностью удовлетворить потребности пользователя.

Список потребностей заинтересованных лиц

В результате анкетирования и интервьюирования всех заинтересованных лиц были сформулированы потребности заказчика относительно разрабатываемого ПО. Далее необходимо провести аналогию между выявленными потребностями и структурой и требованиями ТЗ в соответствии с ГОСТ 34.602–89. Таким образом, потребности заказчика в ТЗ могут быть описаны в разделе «Назначение и цели создания системы».

Допустим, что в нашем примере были выявлены следующие потребности:

1) унифицировать процесс оценивания знаний в системе кредитов на всех кафедрах и факультетах вуза;

2) минимизировать субъективность при оценивании студентов в промежуточных аттестациях;

3) реализовать возможность автоматического формирования рейтингов студентов по разным параметрам;

4) реализовать возможность формирования единой отчетности на кафедрах и факультетах.

Задание и порядок проведения работы

1. Составить перечень заинтересованных лиц.
2. Провести интервью и/или анкетирование с каждым заинтересованным лицом.
3. Проанализировать полученную информацию и сформулировать актуальность проблемы и потребности заинтересованных лиц.

Контрольные вопросы

1. Что является исходными данными для анализа проблемы (предметной области)?
2. Что является результатом этапа системного анализа предметной области?
3. Как определить заинтересованных лиц?
4. Какой метод сбора информации наиболее эффективен?
5. Для чего проводятся интервьюирование и анкетирование?

Работа с литературой:

Рекомендуемые источники информации (№ источника)			
Основная	Дополнительная	Методическая	Интернет-ресурсы
1-3	1-3	1-3	1-2

Оценочные средства: контрольные вопросы

Тема 2. Анализ проблемы и постановка задачи.

Лабораторная работа № 2. Моделирование объекта автоматизации

Форма проведения: Практическое выполнение задания

Цели работы:

- познакомиться с методологией ARIS;
- сформировать навыки работы в среде ARISToolset;
- сформировать навыки разработки организационных диаграмм и диаграмм бизнес-процессов (VAD и eEPC).

Краткие теоретические сведения

На основе собранных данных разрабатывается модель предметной области. Модель – это мысленно представленный, или изображенный (например, нарисованный на бумаге), или изготовленный (например, бумажный макет) образ, аналог оригинала (объекта, процесса или явления), который «замещает» оригинал и отражает его наиболее важные черты и свойства.

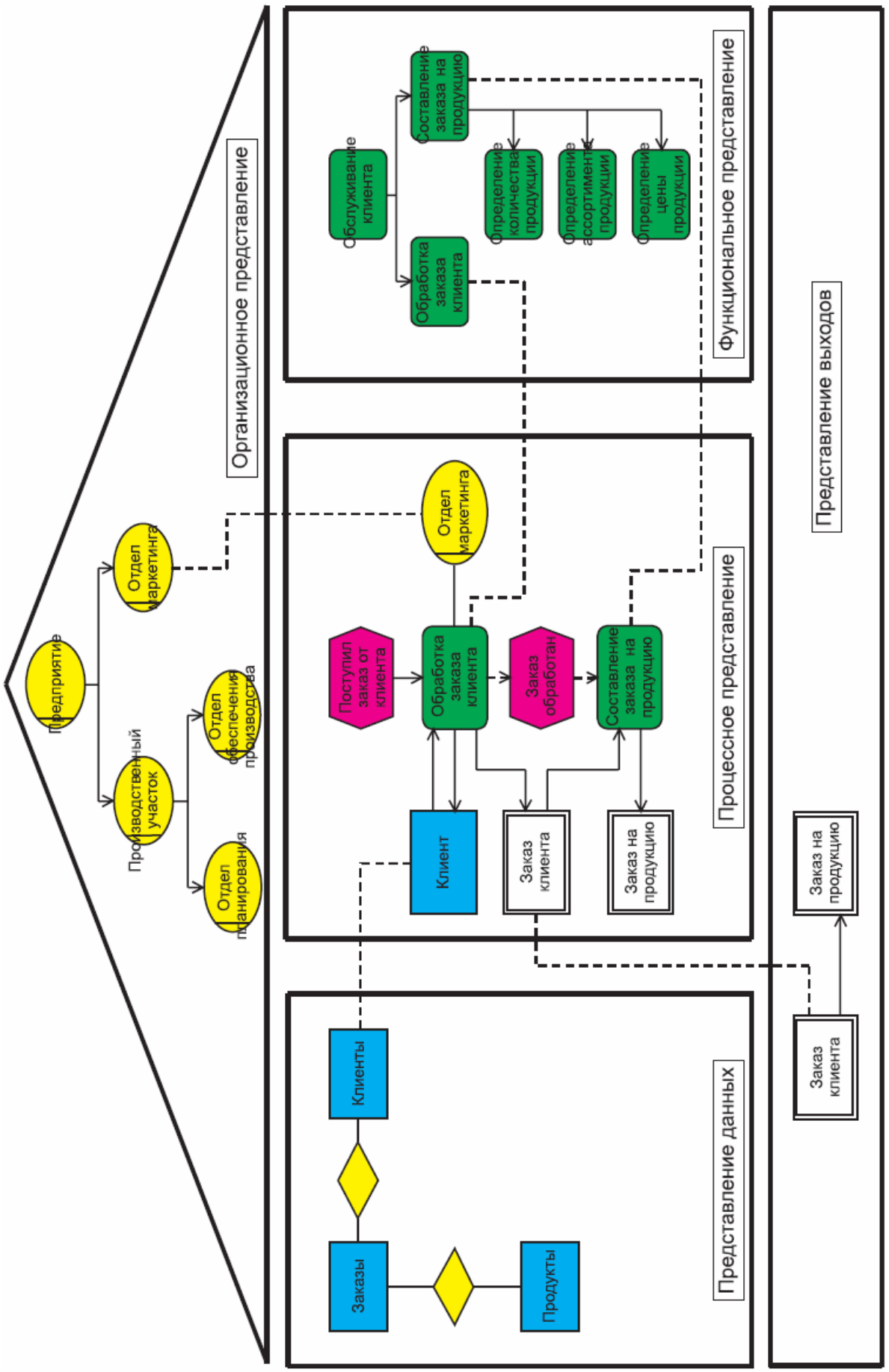


Рисунок 2.1 - «Здание» ARIS. Представления

ARIS-модели – это диаграммы экономических систем и процессов, на основе которых можно анализировать, оптимизировать бизнес-процессы (БП) и управлять ими.

В рамках данной лабораторной работы предлагается строить модель с использованием инструмента ARIS. Выбор этого инструмента и методологии обусловлен заложенными в них принципами, которые основаны на теории систем.

Введение в методологию ARIS

В теории систем различают структуру системы и ее поведение. Структура описывает статику системы, а поведение – ее динамику. Четыре первых представления в ARIS (организационное, функциональное, представление выходов и представление данных) описывают структуру системы. В процессном представлении устанавливаются связи перечисленных представлений и описывается динамическое поведение системы. На рисунке 2.1 перечисленные представления сгруппированы в пять частей, образующих «здание» ARIS.

Функциональное представление содержит описание целей, выполняемых функций, перечень отдельных подфункций, а также общие взаимосвязи и связи подчиненности, которые существуют между функциями, целями и факторами успеха.

Организационное представление описывает взаимодействие пользователей и организационных единиц, а также их связи и имеющие к ним отношение структуры. Организационные модели служат для описания иерархической структуры организации (где группируются субъекты ответственности), организационных единиц, должностей и средств, выполняющих работу над объектами, и др.

В представлении данных описывают информационную среду предприятия (среду обработки данных), события, сообщения и др., где неявно фиксируются также объекты в виде информационных услуг.

Процессное представление используется для описания связей между тремя предшествующими представлениями. Интеграция этих связей в пределах отдельного представления делает возможным учесть все связи без избыточности. Модели представления процессов описывают отношения между отдельными моделями в рамках всего бизнес-процесса. Это позволяет отслеживать все двусторонние и многосторонние отношения между моделями различных видов, а также полностью описать бизнес-процесс.

Представление выходов предназначено для описания результатов выполнения процессов.

Описание инструмента ARIS. Начало работы

Аналогично большинству приложений Windows, ARIS имеет многооконный интерфейс. Главное окно ARIS (рисунок 2.2) состоит из главного меню, панели инструментов, панели проводника (рисунок 2.3) и обзорного окна.

Прежде всего создайте новую базу данных на локальном сервере. Для этого щелкните правой кноПКй мыши по панели проводника на сервере LOCAL – появится контекстное меню. Выберите в нем New→Database.

В появившемся окне введите имя новой базы данных латинскими буквами, например Sample.

Для того чтобы элементы моделей имели русскоязычные имена, добавьте в список языков русский язык. Для этого откройте созданную вами базу данных, выберите папку Languages и через контекстное меню добавьте русский язык. Затем в главном меню выберите View→Options – появится окно настроек ARIS, общих для всех баз данных (рисунок 2.4). На закладке General измените язык English на «Русский» (рисунок 2.4) в поле Database Language.

Перезапустите ARIS.

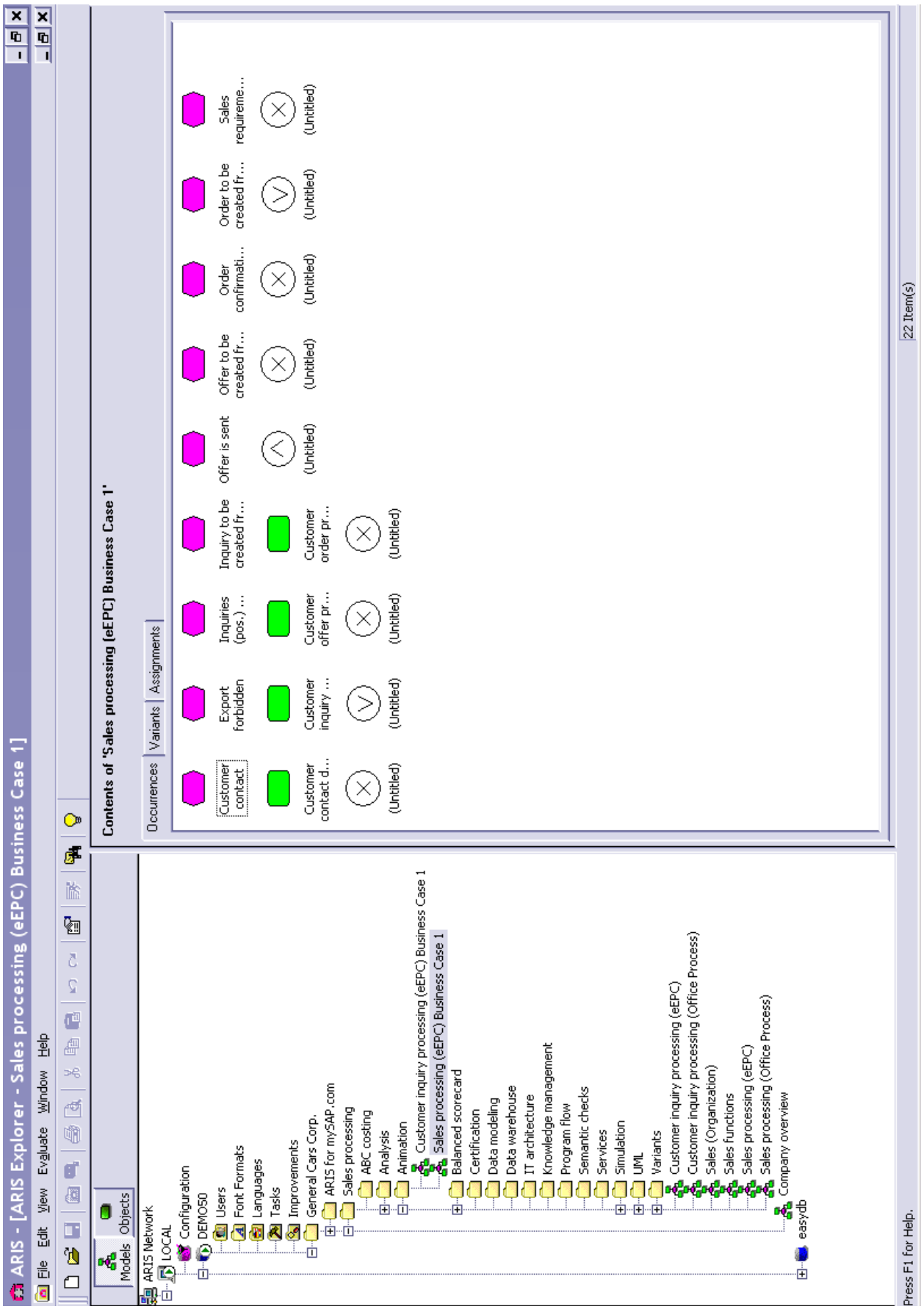


Рисунок 2.2 - Главное окно ARIS

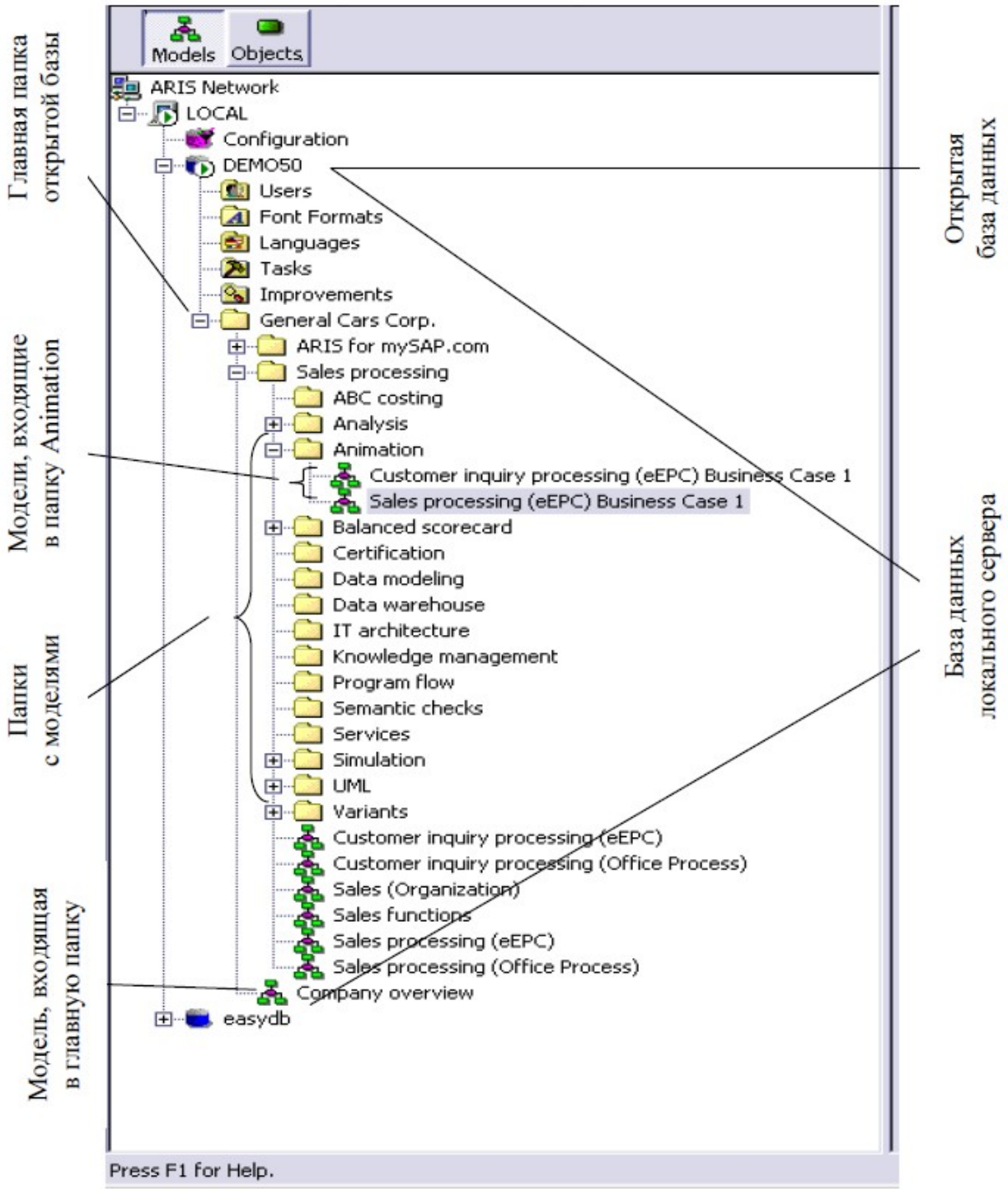


Рисунок 2.3 - Левая панель проводника ARIS

Откройте вашу базу данных, и вы увидите, что главная папка вместо названия имеет неопределенное имя (Untitled). Это означает, что на русском языке эта папка еще не получила названия. Так как ARIS поддерживает многоязычность, то определенные элементы одной и той же модели, а также названия самих моделей и папок могут одновременно иметь несколько названий на разных языках. Это очень удобно в тех случаях, когда выполняются совместные международные проекты, поскольку отобразить модель на другом языке можно нажатием всего нескольких клавиш. Однако очевидно, что названия всех моделей при их разработке должны быть продублированы на всех языках, которые считаются в проекте основными, что увеличивает трудоемкость соответствующих работ по построению моделей.

Измените название главной папки с Untitled, например, на «Главная». Изменить название можно либо через контекстное меню, либо вторым щелчком левой кнопки мыши через временной

промежуток (как в Windows-проводнике). На этом настройку инструмента и базы данных закончим и перейдем к построению моделей или моделированию предметной области.

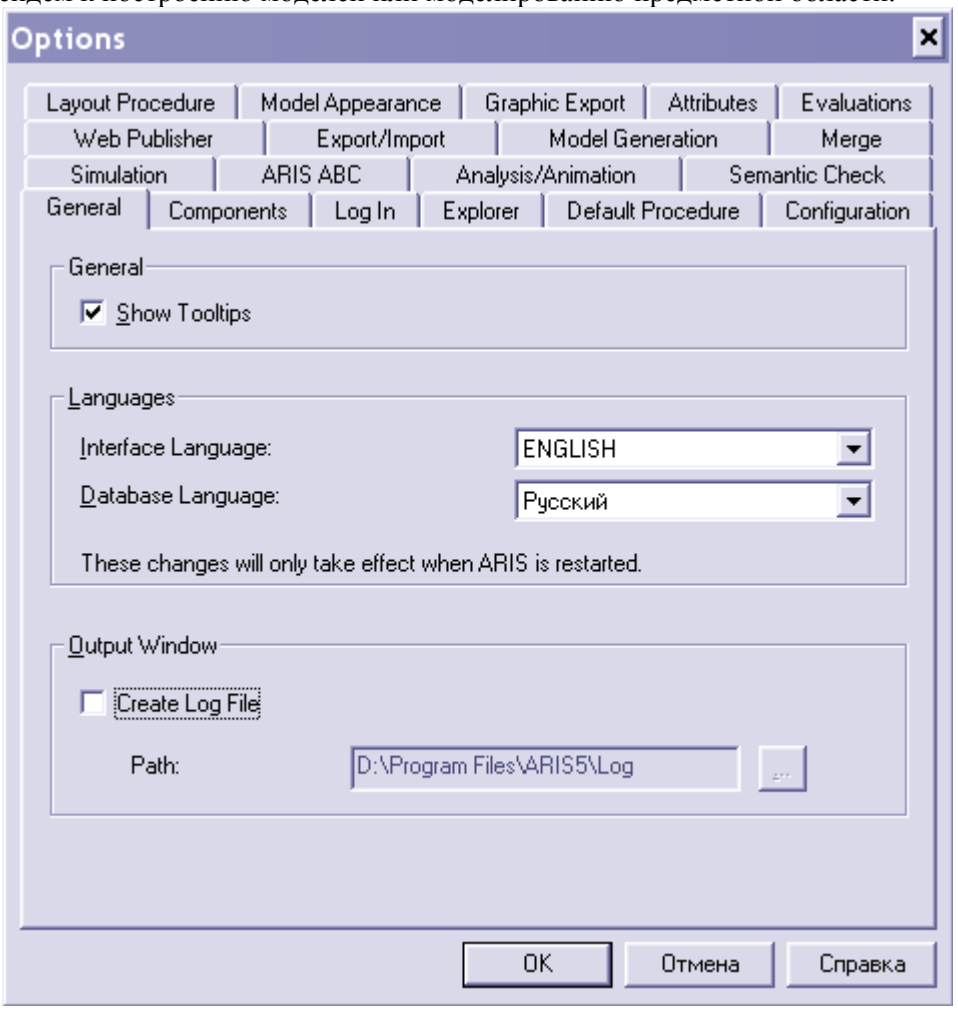


Рисунок 2.4 - Окно настроек

Построение организационной модели

Щелкните правой кноПКй мыши по главной папке и в появившемся контекстном меню выберите New→Model – появится окно выбора модели (рисунок 2.5). Поставьте галочку в поле Organization (рисунок 2.5), что соответствует организационному представлению. Ниже появится список всех моделей, имеющих отношение к выбранному представлению или к выбранным представлениям, если галочками будет помечено несколько полей.

Выберите диаграмму Organization Chart и нажмите кнопку Далее. Введите имя новой модели, например «Организационная структура».

Организационная диаграмма позволяет всесторонне описать организационно-штатную структуру предприятия. Организационные диаграммы показывают имеющиеся организационные подразделения (как исполнители функций) и их взаимозависимости в соответствии с выбранными критериями структурирования. Отдельные организационные единицы соединяются связями для указания иерархии.

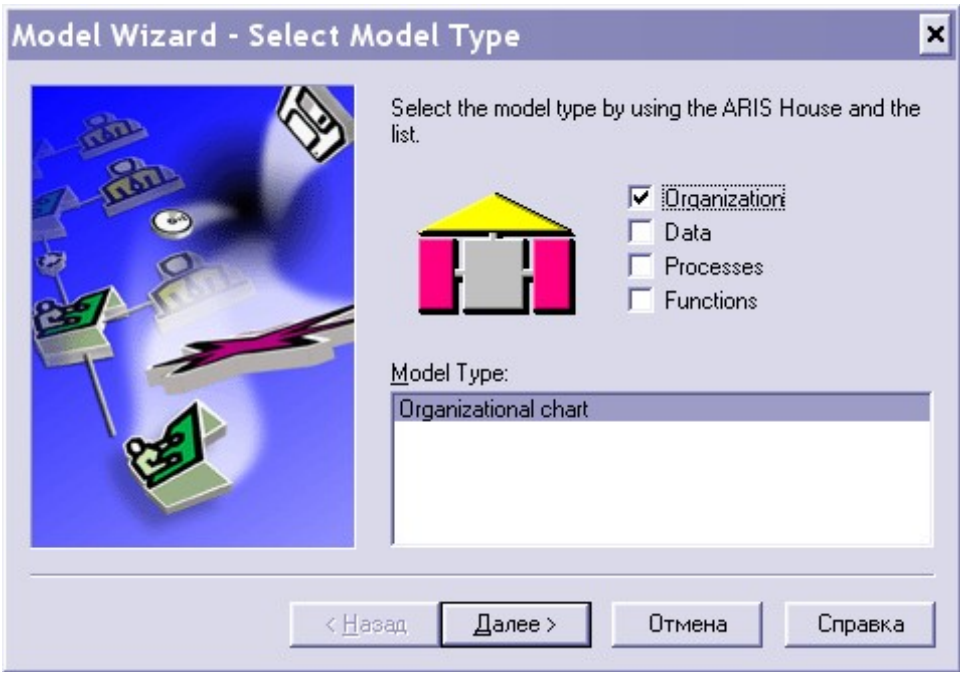
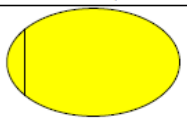
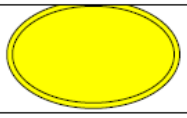
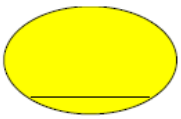


Рисунок 2.5 - Окно создания новой модели

Таблица 2.1 - Элементы организационной диаграммы

Графическая нотация	Наименование	Описание
	Организационная единица (Organizational unit)	Элемент организационной структуры (структурное подразделение), который отвечает за выполнение определенных задач и преследует определенные цели
	Должность или позиция (Position)	Наиболее мелкий организационный элемент на предприятии. Обязанности и административные полномочия такого элемента задаются должностными инструкциями
	Группа (Group)	Несколько сотрудников, которые вместе работают над конкретной задачей в определенный период времени (например, группа проектирования)
	Внешний сотрудник (External person)	Лицо, выполняющее определенные функции в компании, но не являющееся служащим этой компании. Может назначаться к организационным единицам или функциям, к выполнению которых он привлекается или за которые он отвечает
	Сотрудник (Internal person)	Лицо, являющееся служащим компании. Может назначаться к организационным единицам или функциям, которые он выполняет или за которые несет ответственность
	Тип сотрудника (Person type)	Обобщает индивидуальных лиц, имеющих одинаковые характеристики. Эти характеристики могут иметь отношение, например, к одним и тем же полномочиям. Начальники отделов или бригадиры, например, должны следовать определенным правилам и выполнять определенные обязанности, которые достаточно описать только один раз
	Местонахождение (Location)	Определяет физическое положение (область, город, предприятие, здание и т. д.) организационных единиц, рабочих мест, образцов аппаратных компонентов и технических ресурсов компании

Модель организационной структуры предприятия может содержать следующие структурные элементы: Организационная единица, Должность или позиция, Группа и др. (таблица 2.1).

При построении модели организационной структуры между структурными элементами могут устанавливаться следующие типы связей: Имеет в подчинении (Is superior), Связан с (Is assigned to), Принадлежит (Belongs to), Является организационным управляющим (Is Organization Manager for), Имеет в своем составе (Has member), Состоит из (Is composed of), Располагает (Is located at), Взаимодействует с (Cooperates with), Содержит (Subsumes), Занимает (Occupies).

Тип связи устанавливается автоматически после соединения двух элементов модели и зависит от последовательности соединения двух элементов. Для того чтобы посмотреть, какой тип связи установился между элементами, щелкните двойным кликом по связи на диаграмме – отобразится окно свойств этой связи (рисунок 2.6), где в поле Type можно увидеть тип связи.

Пример организационной диаграммы приведен на рис. 2.7.

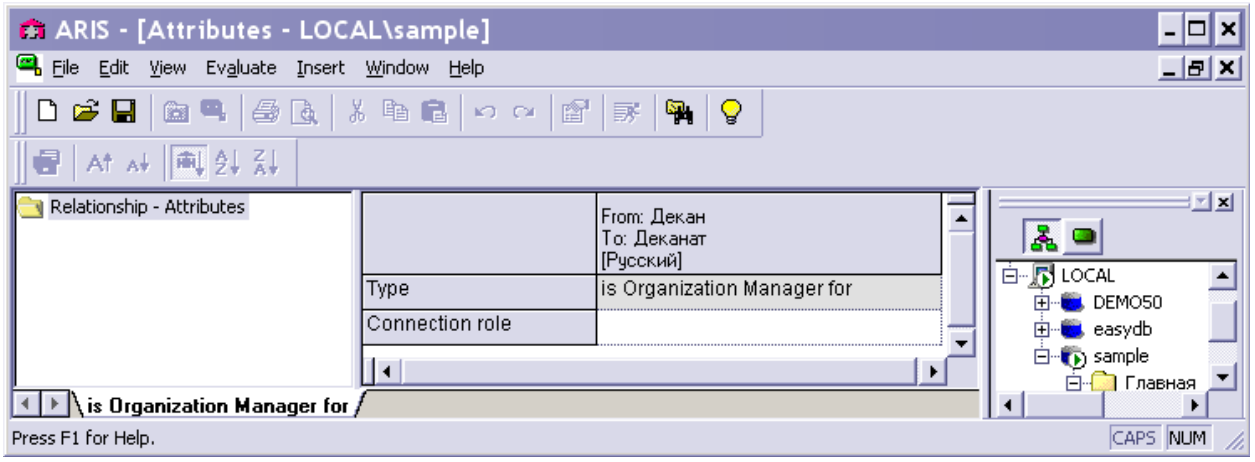


Рисунок 2.6 - Свойства связи между элементами

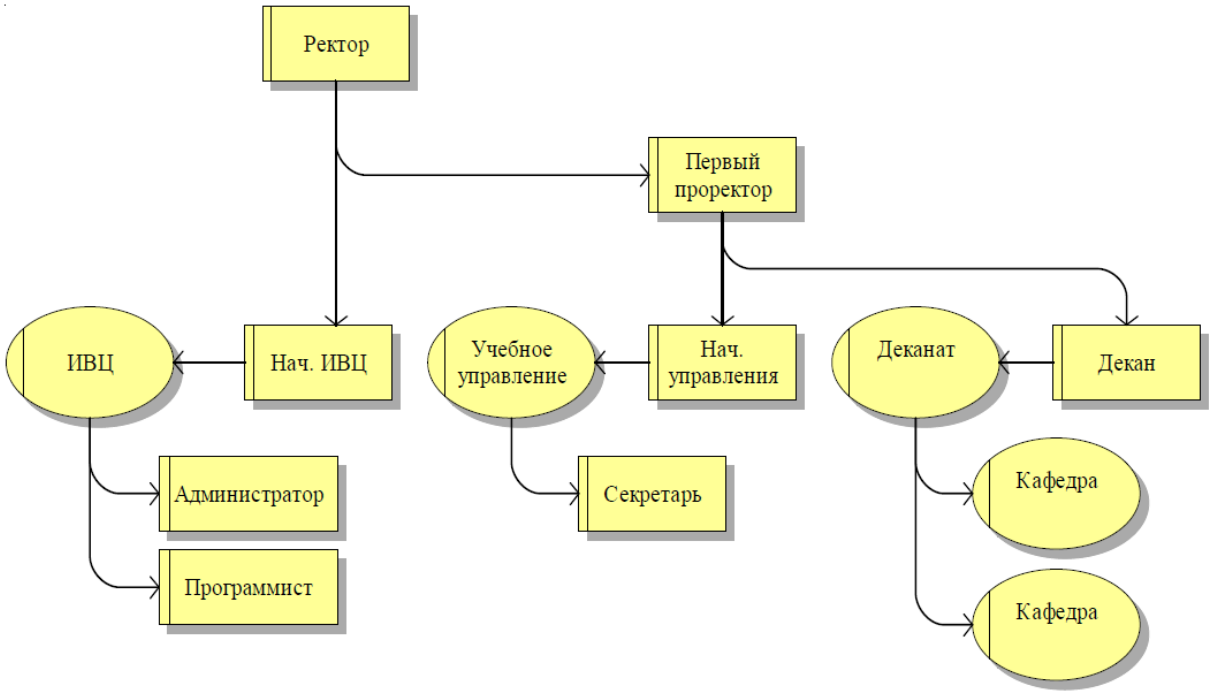


Рисунок 2.7 - Часть организационной диаграммы «Университет»

Построение диаграммы цепочек добавленного качества

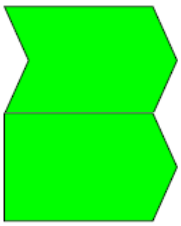
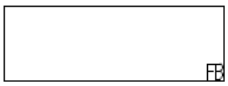
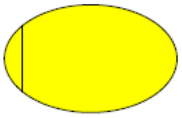
Для упрощения описания деятельности организации необходимо разработать иерархию моделей бизнес-процессов (БП) организации, начиная с самого верхнего уровня и заканчивая моделями отдельных БП на нижнем уровне. Для описания модели верхнего уровня используется диаграмма типа Value-added chain diagram (VAD), название которой можно перевести как «модель цепочки добавленного качества (стоимости)».

Диаграмма цепочек добавленного качества описывает функции организации, которые непосредственно влияют на реальный выход ее продукции. Эти функции создают последовательность действий, формируя добавленные значения: стоимость, количество, качество и т. д.

Для создания диаграммы VAD щелкните правой кноПКй мыши по главной папке и в появившемся контекстном меню выберите New→Model. Появится окно выбора модели (рисунок 2.5). Поставьте галочку в поле Processes, что соответствует процессному представлению.

Выберите диаграмму Value-added chain diagram и нажмите кнопку Далее. Введите имя новой модели, например «Основные процессы». Диаграмма цепочек добавленной стоимости может содержать следующие структурные элементы: Организационная единица, Должность или позиция, Функция и др. (таблица 2.2).

Таблица 2.2 - Элементы диаграммы цепочки добавленного качества

Графическая нотация	Наименование	Описание
	Функция	Действие или набор действий, выполняемых над исходным объектом (документом, материалом и т. п.) с целью получения заданного результата
	Технический термин	–
	Организационная единица (Organizational unit)	Элемент организационной структуры (структурное подразделение), который отвечает за выполнение определенных задач и преследует определенные цели
	Кластер (экземпляр)	Экземпляр объекта данных. Он представляет набор объектов как структуру данных

Пример диаграммы цепочек добавленной стоимости представлен на рисунке 2.8.

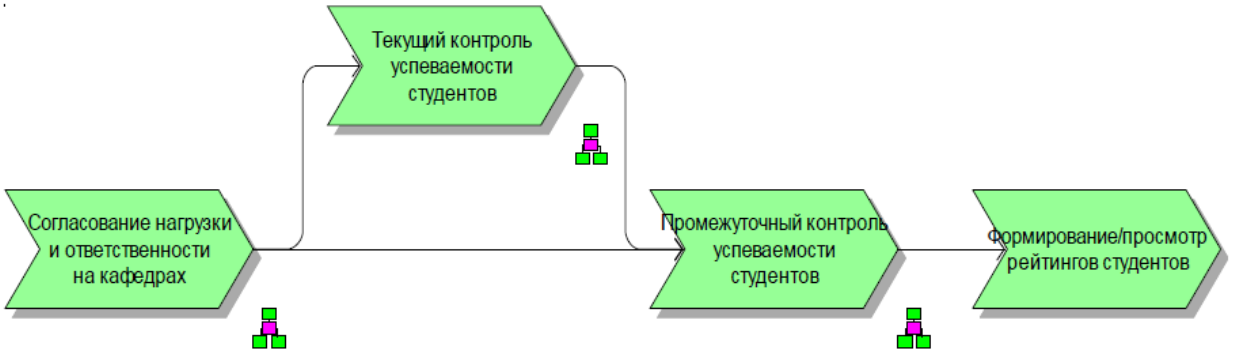


Рисунок 2.8 - Основные процессы учета успеваемости студентов в системе рейтингов

Пиктограммой  помечаются те функции, для которых существуют модели, раскрывающие эту функцию. Таким механизмом реализуется принцип декомпозиции.

Построение eEPC модели

Extended event driven process chain (eEPC) – цепочка процесса, управляемая событиями.

С помощью диаграмм eEPC процедуры БП представляются как логические последовательности событий. События определяют, какое состояние или отношение будет переключать функцию и какое состояние наступит после ее выполнения. Поэтому модель eEPC должна всегда иметь запускающие и завершающие события.

Одно событие может инициировать выполнение одновременно нескольких функций, и, наоборот, функция может быть результатом наступления нескольких событий. Объединения нескольких событий или функций отображаются на диаграмме eEPC с помощью соединителей в виде небольшого кружка. Эти соединители не только отображают графические связи между объектами модели, но и определяют логические связи между ними. Различают два типа связи логических операторов – связи событий и связи функций.

Для создания диаграммы eEPC щелкните правой кноПКй мыши по главной папке и в появившемся контекстном меню выберите New→Model. Появится окно выбора модели (рисунок 2.5). Поставьте галочку в поле Processes, что соответствует процессному представлению.

Выберите диаграмму eEPC и нажмите кнопку Далее. Введите имя новой модели.

Таблица 2.3 - Элементы диаграммы цепочек, управляемых событиями

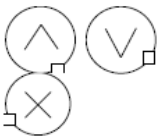
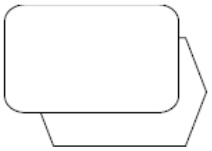
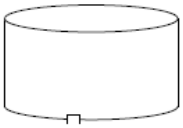
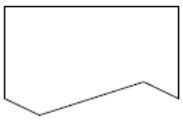
Графическая нотация	Наименование	Описание
	Функция	Действие или набор действий, выполняемых над исходным объектом (документом, материалом и т. п.) с целью получения заданного результата
	Событие	Состояние, которое является существенным для управления БП и оказывает влияние или контролирует дальнейшее развитие одного или более БП. Изменения состояния отражаются с помощью информационных объектов
	Правило	Представляет собой правило разветвления и слияния веток БП. Если перейти к рассмотрению каждой функции БП, то можно сказать, что правило отражает логическое соотношение между несколькими исходными для функции событиями и несколькими результирующими
	Интерфейс функции	Используется для обозначения функции внешнего БП, который либо не описывается в данной модели, либо является БП другой предметной области
	Носитель информации	Представляет собой средство хранения информации. Оно может быть реализовано, к примеру, в виде картотеки или компьютерных файлов или БД
	Документ	Обозначает любой вид документов

Диаграмма цепочек, управляемых событиями, кроме элементов организационной диаграммы и диаграммы данных может содержать следующие структурные элементы: Функция, Событие и др. (таблица 2.3).

При построении диаграмм eEPC нужно соблюдать следующие правила:

1) каждая функция инициируется событием (или несколькими событиями) и завершается событием (или несколькими событиями);

2) каждая функция должна иметь входную и выходную информацию;

3) у каждой функции должно быть ответственное лицо за ее выполнение.

На рисунке 2.9 приведен пример блока, на базе которого строится вся модель eEPC.

Пример диаграмм цепочек, управляемых событиями, представлен на рисунках 2.10 - 2.12. Эти диаграммы раскрывают (детально описывают) процессы, изображенные на рисунке 2.8.

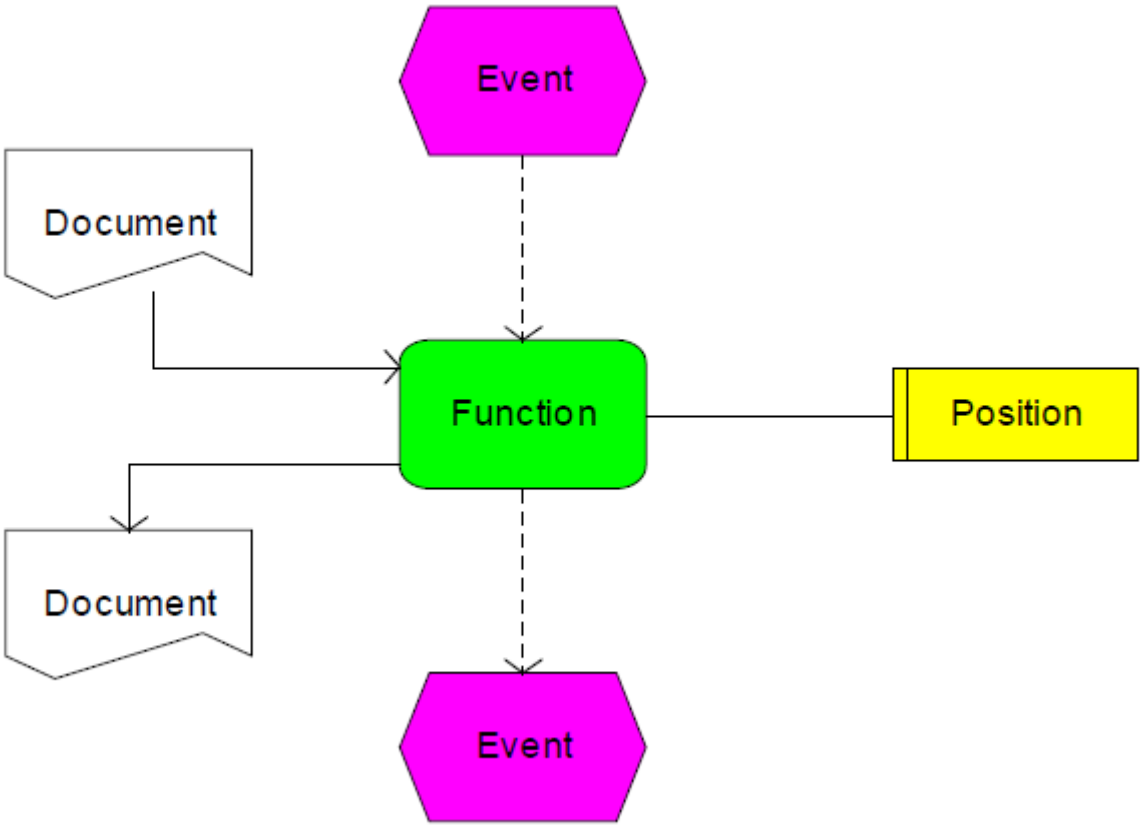


Рисунок 2.9. Блок, на базе которого строится модель eEPC

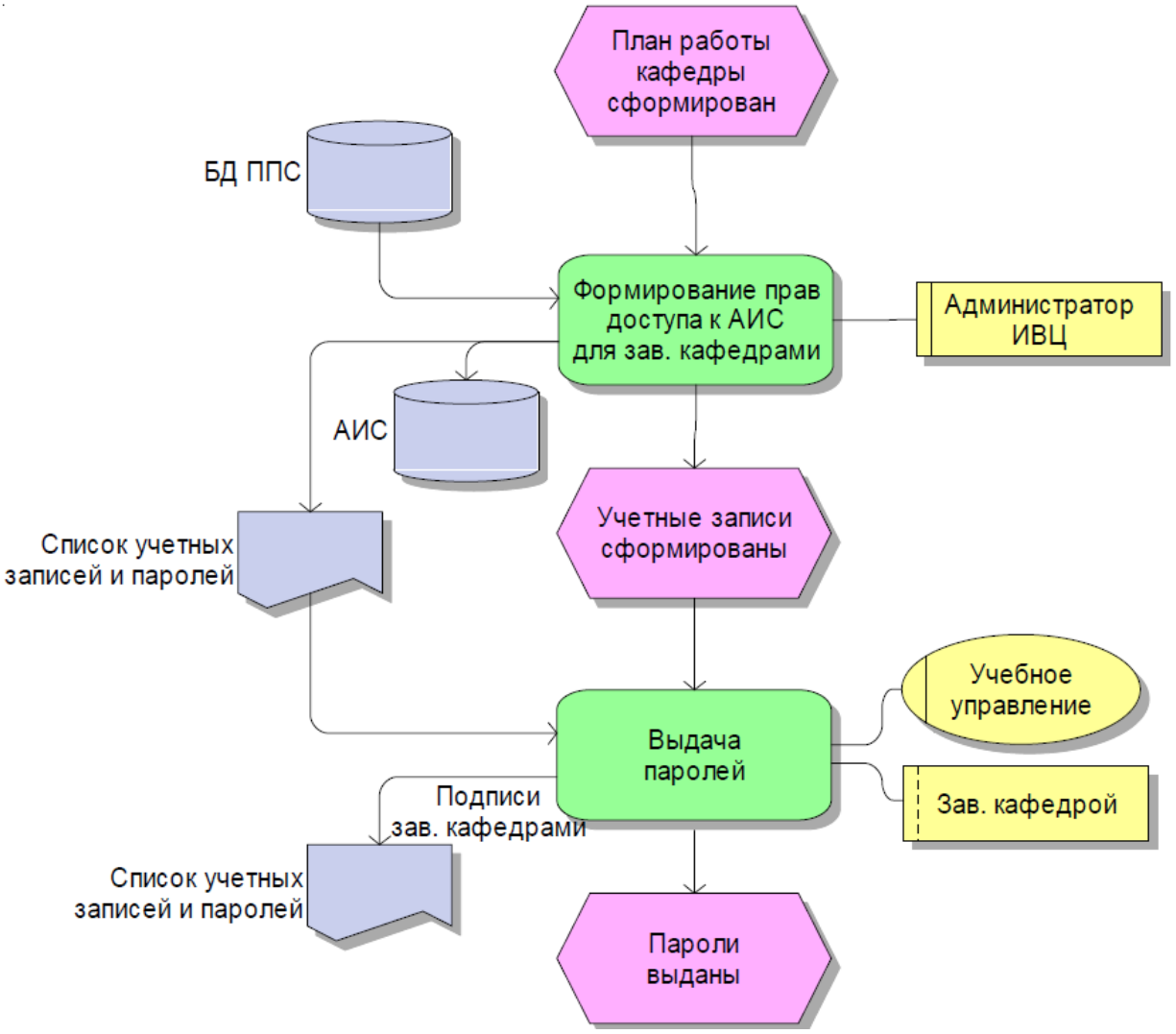
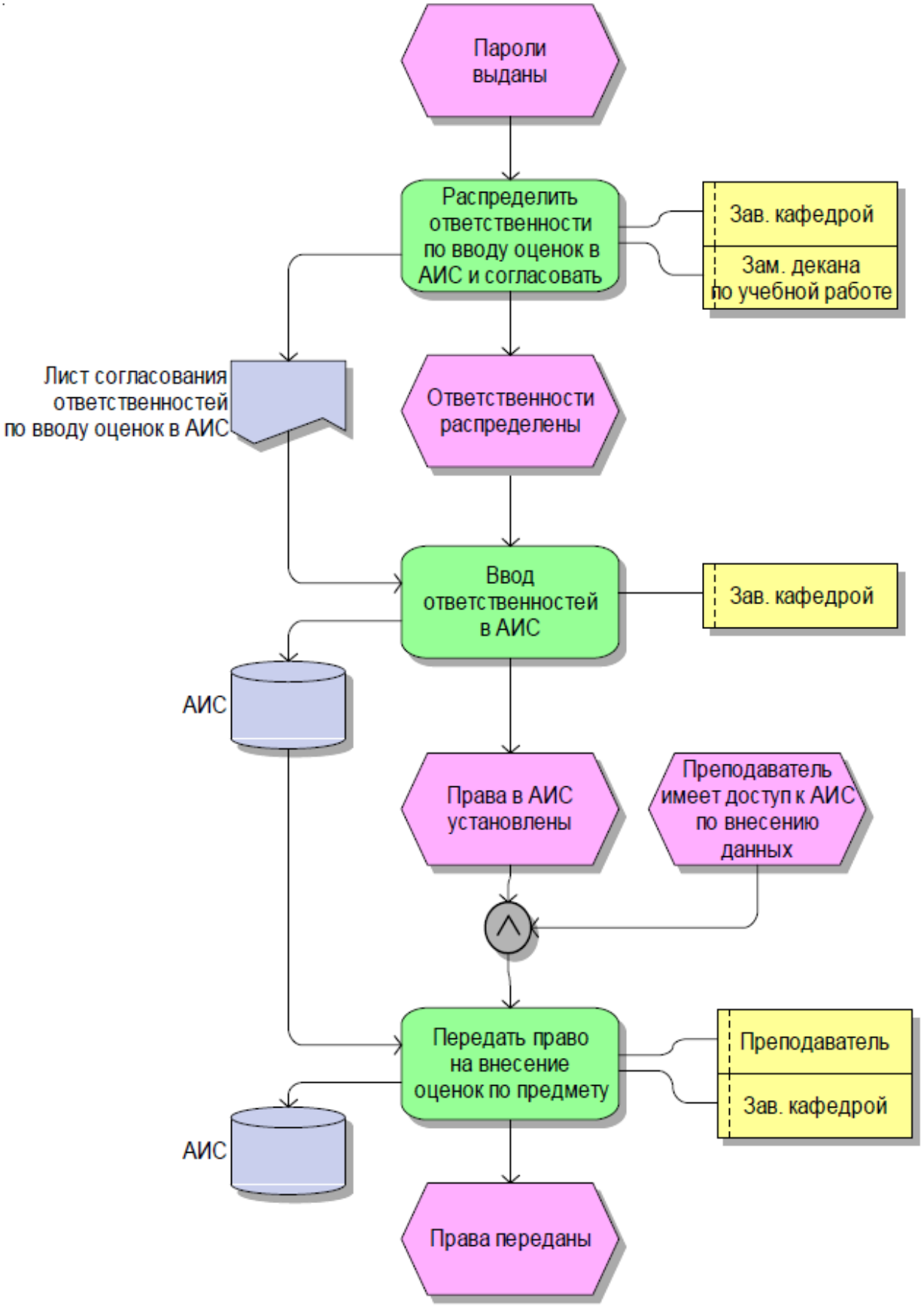


Рисунок 2.10 - Согласование нагрузки и ответственности на кафедрах



Продолжение рисунка 2.10 – Согласование нагрузки и ответственности на кафедрах

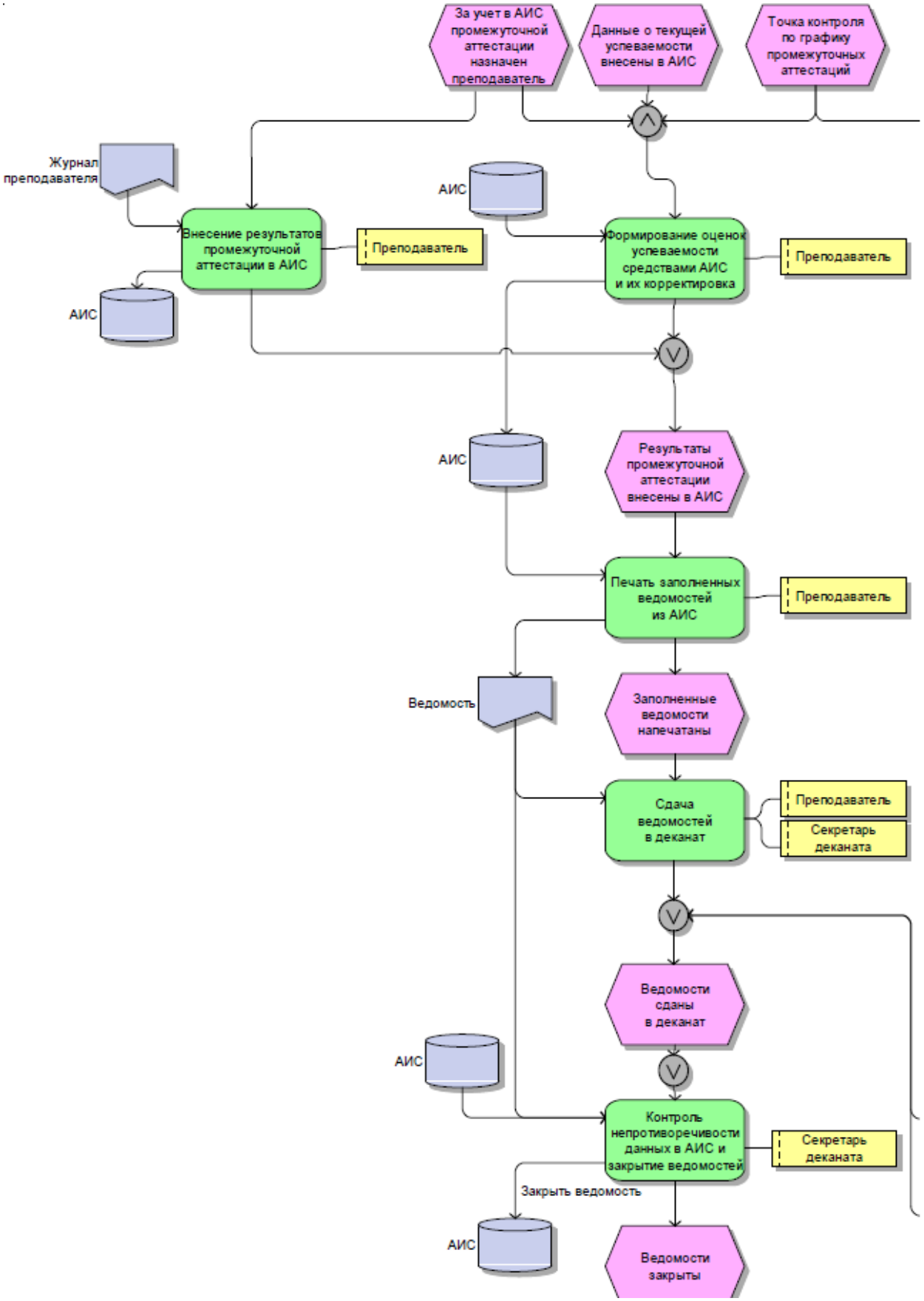
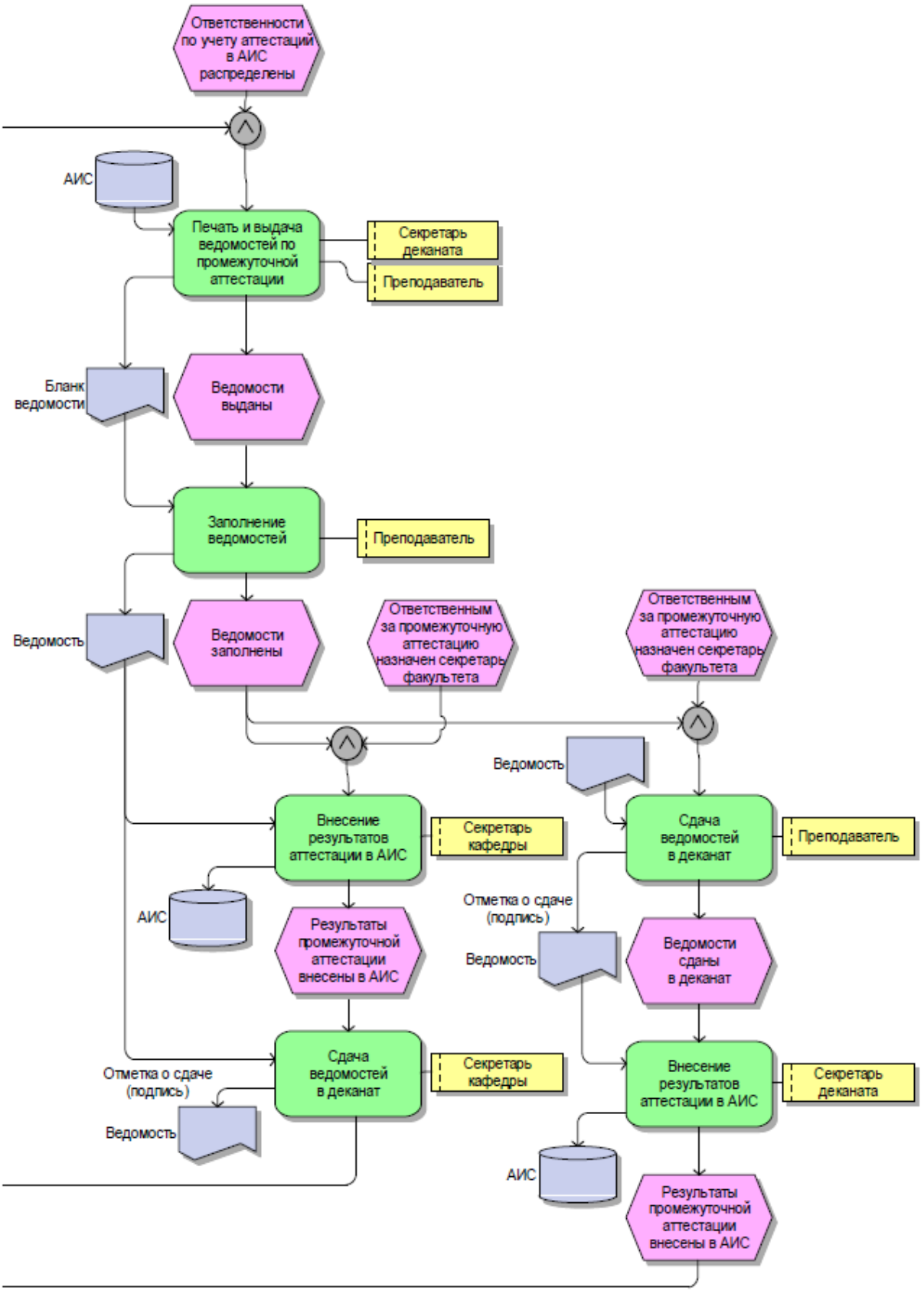


Рисунок 2.11. Промежуточный контроль успеваемости студентов



Продолжение рисунка 2.11 - Промежуточный контроль успеваемости студентов

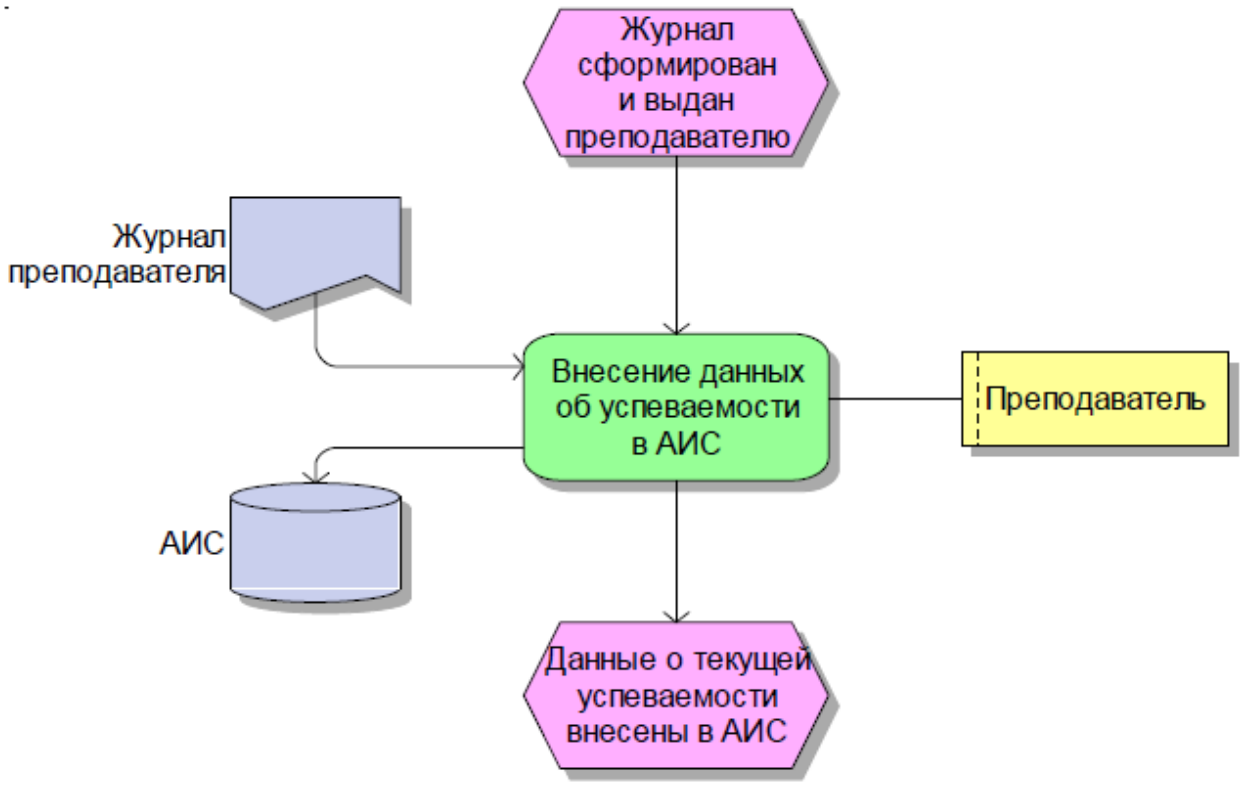


Рисунок 2.12. Текущий контроль успеваемости студентов

Описание объектов автоматизации

Объект автоматизации необходимо описывать с целью структурирования полученных данных об этом объекте и представления их в письменном и/или модельном виде. Представленная таким образом информация об объекте является незаменимым помощником при выполнении последующих работ в процессе разработки ПО, таких как проектирование подсистем и модулей, проектирование базы данных, разработка пользовательского интерфейса, формулирование функций и требований к ПО. Кроме того, описание объекта автоматизации требует ГОСТ при разработке ТЗ.

Как правило, для объекта строится его модель (например, по методологии ARIS), делаются пояснения к модели и описываются нюансы, которые сложно или невозможно отобразить на модели.

Задания и порядок выполнения работы

1. Установить и настроить среду моделирования ARISToolset.
2. Разработать организационную диаграмму.
3. Разработать диаграмму добавленного качества (VAD).
4. Разработать для каждого процесса VAD диаграмму eEPC.

Контрольные вопросы

1. Что такое модель?
2. Методология ARIS.
3. Организация хранения данных о модели в ARISToolset.
4. Как взаимосвязаны разные диаграммы в рамках одной модели?
5. Какие вы знаете связи между элементами в организационной диаграмме?
6. Какие процессы отражены на диаграмме VAD?
7. Как реализован механизм декомпозиции в ARIS?
8. Из каких блоков строится диаграмма eEPC?
9. Что такое событие в eEPC?
10. Каким образом может использоваться модель ARIS для описания объекта автоматизации?

Работа с литературой:

Рекомендуемые источники информации (№ источника)

Основная	Дополнительная	Методическая	Интернет-ресурсы
1-3	1-3	1-3	1-2

Оценочные средства: контрольные вопросы

Тема 3. Анализ требований и их формализация.

Лабораторная работа № 3. Разработка модели вариантов использования и их спецификаций

Форма проведения: Практическое выполнение задания

Цель работы – сформировать навыки:

- разработки модели вариантов использования;
- организации модели вариантов использования;
- разработки спецификации вариантов использования.

Краткие теоретические сведения

Цель данного этапа заключается в разработке требований к ПО. В России стандартом оформления требований является ГОСТ 34.602–89, который регламентирует содержание ТЗ на разработку автоматизированных информационных систем.

Существуют и другие широко применяемые в мире стандарты, например IEEE (Institute of Electrical and Electronics Engineers) 830: Standard for Software Requirements Specification (1994).

Процесс разработки требований к ПО (или спецификаций требований) условно можно разделить на три этапа, которые позволяют описать требования на трех уровнях (рисунок 3.1), причем настолько детально, насколько этого требует уровень абстракции.

Рисунок 3.1 - Характеристика уровней описания требований

Выявление и формулирование требований первого уровня «Потребности» было осуществлено на предыдущем этапе – этапе системного анализа. На настоящем этапе нужно описать требования для второго и третьего уровней.

Модель вариантов использования

Для определения (выявления) функций разрабатываемой системы необходимо провести анализ требований. На данном этапе для такого анализа имеется уже достаточно много материала, который был собран на предыдущем этапе. Таким образом, исходными данными для анализа требований являются потребности заинтересованных лиц, данные интервьюирования, описание объектов автоматизации, цели и задачи разработки системы.

Процесс формулирования функций системы можно частично отнести к процессу проектирования, поскольку он определяет функциональность будущей системы, что, в свою

очередь, уже на этапе анализа требований заставляет задумываться об архитектуре системы и об элементах пользовательского интерфейса. И этот подход верен: он позволяет безболезненно перейти от этапа к этапу и поддерживает итеративную модель разработки программных систем.

При определении функций можно использовать элементы системного анализа. Но существуют методы более адаптированные к процессам разработки ПО. В данной лабораторной работе рекомендуется использовать метод вариантов использования для идентификации и представления функций системы.

Метод вариантов использования (*прецедентов*) является частью методологии объектно-ориентированного проектирования. Это метод анализа и проектирования сложных систем, представляющий собой способ описания поведения системы с точки зрения того, как различные пользователи взаимодействуют с ней для достижения своих целей. Такой ориентированный на пользователя подход предоставляет возможность исследовать различные варианты поведения системы при раннем привлечении пользователя.

Варианты использования можно успешно применять на протяжении всего жизненного цикла ПО: при анализе, проектировании и в процессе тестирования. В этом случае процесс разработки ПО называют «основанный на вариантах использования».

Вариант использования – это функциональный связный блок, выраженный в виде транзакции между актантом и системой (рисунок 3.2). Вариант использования описывает последовательность действий, выполняемых системой с целью получения полезного результата пользователем системы.

Актантом называют пользователя системы (человек, другая система). Между вариантами использования и актантами существует связь, которая показывает, какие функции системы доступны каждому пользователю.

Кроме того, в модели вариантов использования есть связи между самими вариантами использования. В соответствии со стандартом языка UML таких связей достаточно много.

В данной работе будут рассмотрены наиболее часто употребляемые связи при построении модели – это расширение (обобщение) и использование (рисунок 3.2).

Расширение применяется в случае, если у ряда вариантов использования есть общая часть, которую можно выделить в отдельный вариант использования. Этот тип связи аналогичен наследованию в классах. Пример такого типа связи приведен на рисуноке 3.3.

Использование применяется в случае, если несколько вариантов использования в основном или альтернативных потоках имеют одинаковое поведение. В отличие от обобщения использование не подразумевает обязательное наличие общего поведения.

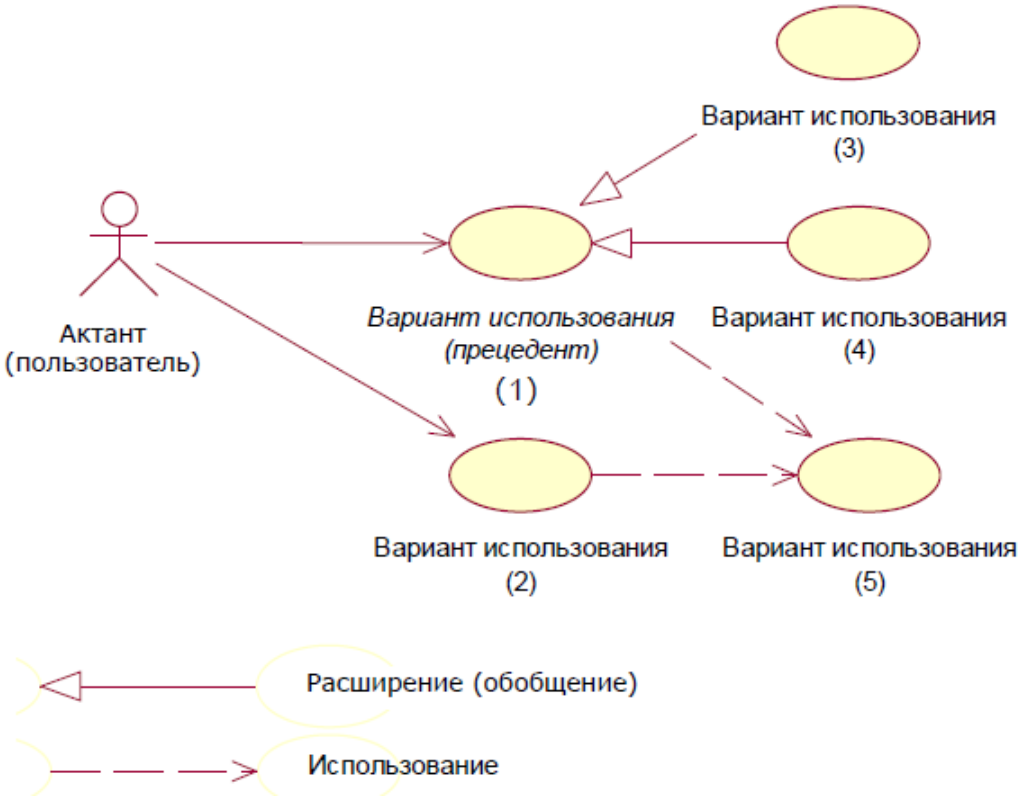


Рисунок 3.2 - Обозначения в модели вариантов использования

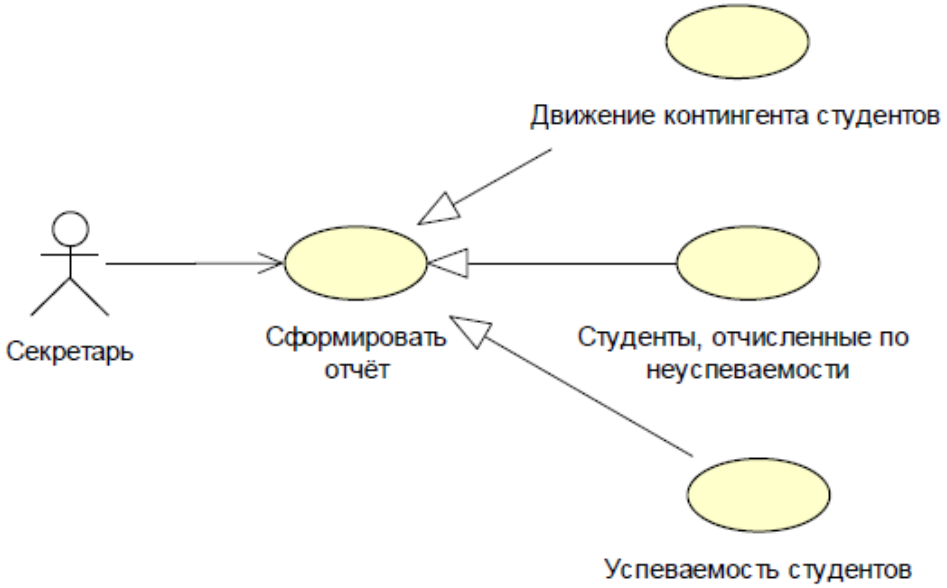


Рисунок 3.3 - Пример связи обобщения

Построение модели вариантов использования.

Модель вариантов использования системы состоит из всех актантов системы и различных вариантов использования, посредством которых актанты взаимодействуют с системой, тем самым описывая многообразие ее функционального поведения. Она также показывает связи между вариантами использования, что углубляет наше понимание системы.

Первый шаг моделирования вариантов использования состоит в создании списка актантов системы. Второй шаг описывает системное поведение, т. е. то, как пользователи взаимодействуют с системой, осуществляя некоторые последовательности действий, для достижения определенных

целей. Если модель вариантов использования получается слишком громоздкой и нечитабельной, то ее необходимо декомпозировать, т. е. разбить на подсистемы.

Пример модели вариантов использования приведен на рисунках 3.4 - 3.8.

Следующий шаг состоит в уточнении деталей функционального поведения каждого варианта использования, т. е. в разработке спецификаций. Спецификации вариантов использования состоят из текстовых и/или графических описаний каждого варианта использования, написанных с позиции пользователя.

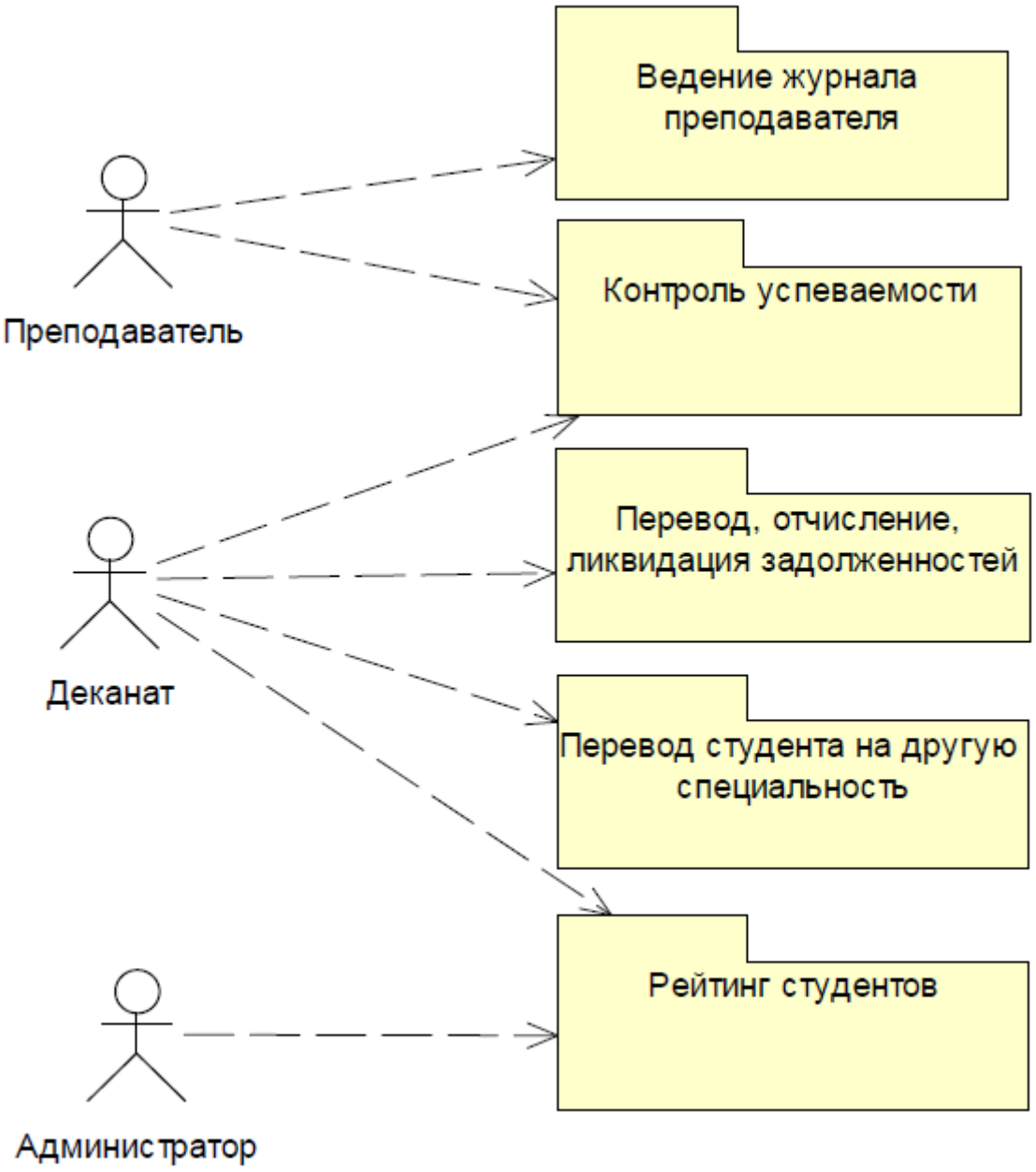


Рисунок 3.4 - Подсистемы в модели вариантов использования

Рисунок 3.5 - Диаграмма вариантов использования «Ведение журнала преподавателя»

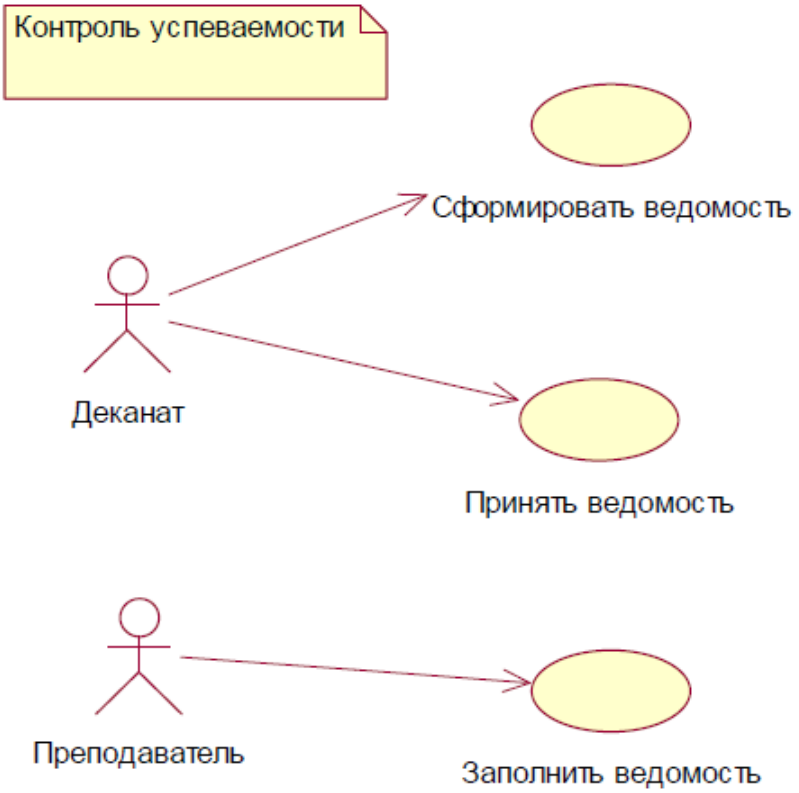


Рисунок 3.6 - Диаграмма вариантов использования «Контроль успеваемости»

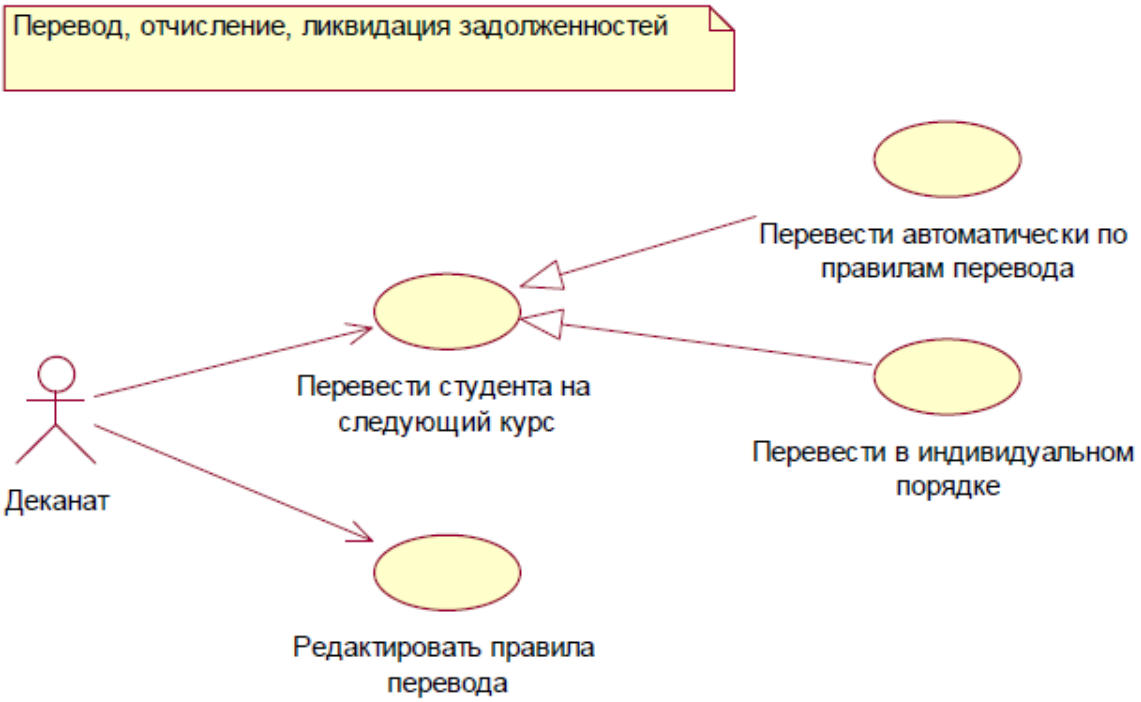


Рисунок 3.7 - Диаграмма вариантов использования «Перевод, отчисление, ликвидация задолженностей»

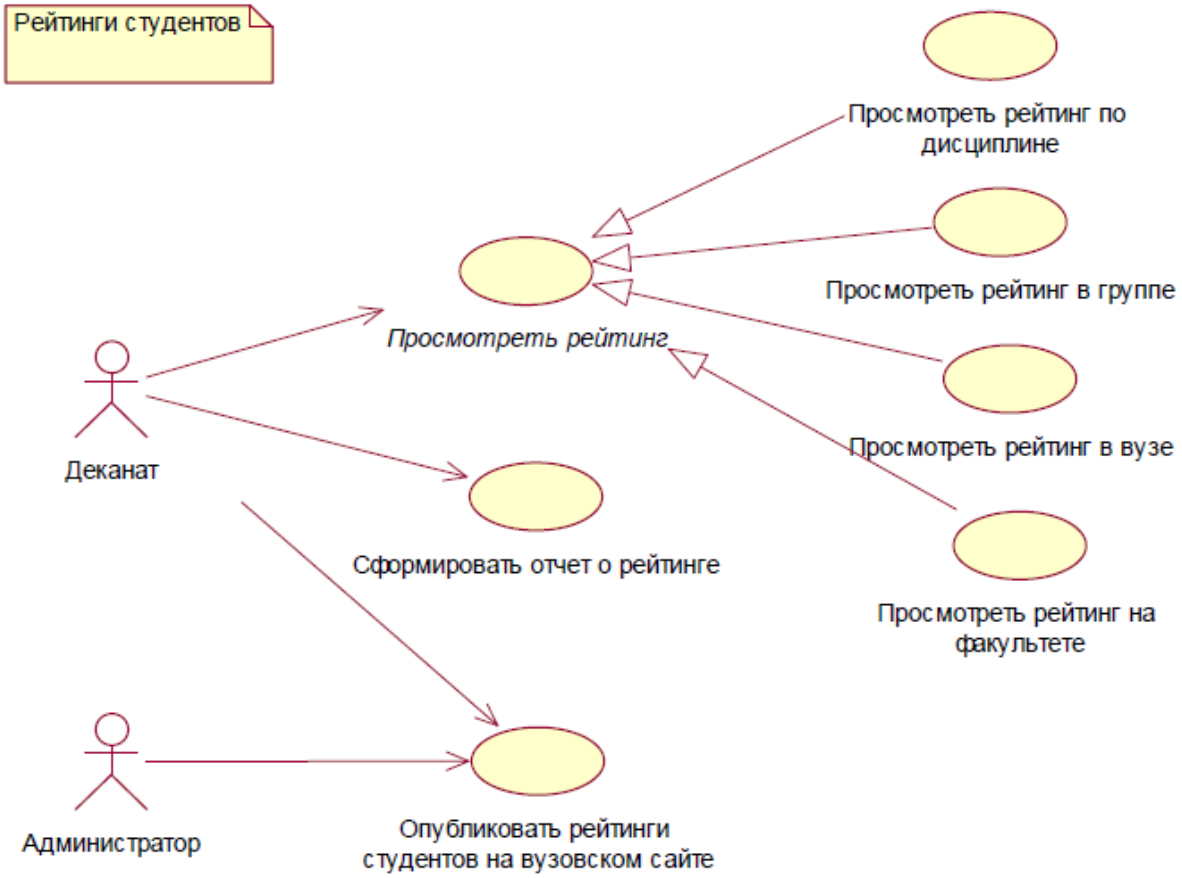


Рисунок 3.8 - Диаграмма вариантов использования «Рейтинги студентов»

Спецификация вариантов использования

Как было отмечено выше, спецификация служит для описания последовательности действий пользователя и системы в рамках одного варианта использования. Цель разработки спецификаций заключается в описании требований к ПО, которые позволят заказчику и разработчику системы однозначно (недвусмысленно) понимать требования. Спецификации вариантов использования позволяют проектировщикам и программистам без дополнительных уточнений и домысливания разрабатывать систему.

Спецификации вариантов использования можно оформлять, придерживаясь разных стандартов, начиная с международных и заканчивая корпоративными. Спецификация должна включать следующие разделы:

- название варианта использования;
- краткое описание варианта использования;
- его роль и цель (для описания достаточно одного абзаца);
- основной поток;
- альтернативные потоки;
- специальные требования.

Основной поток. Вариант использования всегда инициируется неким актантом, когда последний производит некое действие. Вариант использования должен описывать, что делает актант и что система делает в ответ; он должен выглядеть как диалог между актантом и системой.

Вариант использования должен описывать, что происходит внутри системы, а не как или почему это происходит. Если происходит обмен информацией, то нужно указать, какая информация поступает, а какая – отправляется.

Кроме того, не очень понятно, что имеется в виду, если сказать, что актант вводит информацию о клиенте; лучше сказать, что актант вводит имя и адрес клиента. Для того чтобы не делать вариант использования очень сложным и не запутаться в деталях, полезно использовать глоссарий, где можно определить, что понимается под информацией о клиенте.

Простые альтернативы можно описать в тексте варианта использования. Если для описания того, что происходит в альтернативном случае, достаточно нескольких предложений, следует сделать это непосредственно в разделе, посвященном основному потоку событий. Если альтернативные потоки более сложные, нужно использовать отдельный раздел «Альтернативные потоки».

Иногда рисунок информативнее многих слов. Если это поможет добиться большей ясности, можно включать в вариант использования графические описания интерфейсов пользователя, потоков процессов или другие рисунки. Если для представления сложного процесса принятия решения необходимо использовать формальные средства спецификации, такие как диаграммы деятельности, несомненно, это следует делать. Аналогично, если поведение системы зависит от состояния, то диаграмма перехода состояний лучше прояснит его, чем это сделают страницы текста. Используйте представление, которое лучше всего подходит для отображения вашей проблемы, но будьте осторожны с терминологией, обозначениями или рисунками, которые могут быть непонятны вашей аудитории. Помните, что ваша задача прояснить, а не запутать.

Альтернативные потоки

Первый альтернативный поток. Более сложные альтернативы нужно описывать в отдельном разделе. Следует воспринимать альтернативные потоки как варианты альтернативного поведения: каждый альтернативный поток представляет некое альтернативное поведение (вариантов много из-за исключительных ситуаций, возникающих в основном потоке). Они могут быть произвольной длины, которая требуется для описания связанных с альтернативным поведением событий. Когда альтернативный поток заканчивается, события основного потока продолжаются, если не оговорено противное.

Альтернативные потоки могут, в свою очередь, также состоять из подразделов.

Второй альтернативный поток. Достаточно часто в варианте использования встречается несколько альтернативных потоков. Для большей ясности следует описывать каждую альтернативу отдельно. Использование альтернативных потоков упрощает понимание варианта использования, а также предотвращает декомпозицию вариантов использования на их иерархии. Следует помнить, что варианты использования – это только текстовые описания, и их главная задача в том, чтобы документировать поведение системы ясно, сжато и понятно.

Специальные требования. Здесь обычно приводятся имеющие отношение к данному варианту использования нефункциональные требования, которые сложно описать в тексте потока событий варианта использования.

Примерами таких требований могут служить положения законодательства и инструкций, применяемые стандарты, атрибуты качества создаваемой системы, в том числе требования практичности, надежности, производительности или возможности сопровождения. В данном разделе следует также фиксировать другие требования, такие как описание операционных систем и сред, требования совместимости и ограничения проектирования.

Специальные требования включают *предварительные условия варианта использования* (состояние системы, в котором она должна находиться перед началом выполнения варианта использования) и *постусловия варианта использования* (перечень возможных состояний системы непосредственно после завершения варианта использования).

Пример спецификации варианта использования.

Пример приводится для варианта использования «Сформировать отчет о рейтинге».

Краткое описание.

Отчет показывает рейтинги студентов, дисциплин, специальностей, кафедр и факультетов по семестрам, годам и за все время обучения. Перед формированием отчета может быть установлен фильтр для уменьшения полезной (необходимой в данный момент) информации.

Предварительные условия.

Все данные о студентах и результатах аттестаций должны быть внесены в базу данных.

Поток событий. Основной поток.

Пользователь выбирает пункт меню – кнопку, ссылку на усмотрение проектировщика интерфейса пользователя. Система запускает форму выбора параметров отчета «Рейтинги студентов». На этой форме должны быть доступны для выбора следующие параметры:

1. *Дата формирования отчета*. Отчет может быть сформирован за любую дату, но не позднее текущей. Соответственно данные, используемые для формирования отчета, должны соответствовать указанной дате. Это означает, что при выполнении запроса к БД необходимо учитывать даты создания обрабатываемых документов, которые не должны превышать значения, указанного в данном параметре.

2. *Факультет*. Пользователь должен иметь возможность выбрать любой факультет или все факультеты.

3. *Кафедра*. Пользователь должен иметь возможность выбрать любую кафедру или все кафедры. Если в предыдущем пункте был выбран конкретный факультет, то в список кафедр должны попасть только те, которые прикреплены к выбранному факультету.

4. *Направление, специальность, специализация*. Правило выбора аналогично п. 3.

5. *Учебная группа*. Правило выбора аналогично п. 3.

6. *Студент*. Правило выбора аналогично п. 3.

7. *Результаты всех сессий* на дату, указанную в п. 1.

8. *Учебный год*. Список параметров должен формироваться на основе выбранных в п. 1–6 параметров.

9. *Результаты сессии*. Выбирается номер семестра. Если выбран параметр п. 8, то в списке семестров должны находиться только те, которые попадают в выбранный учебный год.

10. *Результаты аттестаций*. В список аттестаций должны попасть только те, которые соответствуют параметрам, выбранным в п. 7, 8.

После выбора на форме соответствующих параметров пользователь нажимает кнопку Сформировать отчет. Система формирует отчет в формате MS Excel в соответствии с формой, приведенной в таблицей 3.1.

Таблица 3.1 - Отчет по рейтингам

Объект	Общий рейтинг	1-й курс						2-й курс						3-й курс					
		1-й семестр			2-й семестр			3-й семестр			4-й семестр			5-й семестр			6-й семестр		
		1 ат.	2 ат.	3 ат.	1 ат.	2 ат.	3 ат.	1 ат.	2 ат.	3 ат.	1 ат.	2 ат.	3 ат.	1 ат.	2 ат.	3 ат.	1 ат.	2 ат.	3 ат.
Факультет1																			
Кафедра1																			
Специальность																			
Группа1																			
ФИО студента1																			
ФИО студента2																			
…																			
Группа2																			
…																			
Специальность																			
…																			
Кафедра2																			
…																			
Факультет2																			
…																			

В этой таблице представлен отчет за первые три курса, 4-й и 5-й курсы добавляются с правой стороны отчета по аналогии с 1–3-м курсами. В зависимости от выбора параметров в п. 1–9 форма отчета может изменять свое содержание, как по вертикали, так и по горизонтали. Рейтинг для каждой ячейки отчета рассчитывается по алгоритму, описываемому в следующем разделе.

Алгоритм расчета рейтингов

В таблице 3.2 приведены обозначения, используемые в формулах по расчету рейтингов.

Трудоемкость поддисциплины по видам занятий (без учета форм отчетности) рассчитывается по формуле

$$T^{p}_{i_{jk},pd_d} = \sum_{t,m_{pd_d}} T^{p,t}_{i_{jk},m_{pd_d}}.$$

Трудоемкость поддисциплины в целом рассчитывается с учетом формы отчетности по формуле

$$T_{i_{jk},pd_d} = \sum_{p} T^{p}_{i_{jk},pd_d} + T^{\text{фо}}_{i_{jk},pd_d}.$$

Средневзвешенная оценка для каждого студента по модулю за каждую аттестацию и по поддисциплине рассчитывается по нижеприведенным формулам:

$$O^{t,m_{pd_d}}_{l,f,sg_{i_{jk}}} = \frac{\sum_{p} O^{p,t,m_{pd_d}}_{l,f,sg_{i_{jk}}} T^{p,t}_{i_{jk},m_{pd_d}}}{\sum_{p} T^{p,t}_{i_{jk},m_{pd_d}}};$$

$$O^{pd_d}_{l,f,sg_{i_{jk}}} = \frac{\sum_{t,m_{pd_d}} \sum_{p} O^{p,t,m_{pd_d}}_{l,f,sg_{i_{jk}}} T^{p,t}_{i_{jk},m_{pd_d}} + O^{pd_d}_{l,f,sg_{i_{jk}}} T^{\text{фо}}_{i_{jk},pd_d}}{\sum_{t,m_{pd_d}} \sum_{p} T^{p,t}_{i_{jk},m_{pd_d}} + T^{\text{фо}}_{i_{jk},pd_d}}.$$

Соответственно рейтинги студентов по модулям за текущие аттестации и рейтинги студентов по поддисциплине рассчитываются по формулам

$$R_{l,f,s_{g_{i_{jk}}}}^{t,m_{pd_d}} = O_{l,f,s_{g_{i_{jk}}}}^{t,m_{pd_d}} \; T_{i_{jk},m_{pd_d}}^{t},$$

$$R_{l,f,s_{g_{i_{jk}}}}^{pd_d} = O_{l,f,s_{g_{i_{jk}}}}^{pd_d} \left(\sum_t T_{i_{jk},m_{pd_d}}^{t} + T_{i_{jk},pd_d}^{\text{фо}} \right).$$

Таблица 3.2 – Обозначения формул расчета рейтинга

Обозначения	Пояснения
$O_{l,f,s_{g_{i_{jk}}}}^{p,t,m_{pd_d}}$	Оценка за текущую работу в 100-балльной шкале, где: p – вид занятий (лекции, лабораторные, экзамен и т. д.); t – номер аттестации в семестре; m_{pd_d} – модуль поддисциплины pd дисциплины d; l – форма оплаты (бюджетники, контрактники); f – категории обучаемых (типы: студент, аспирант и т. д.) и формы обучения (очная, заочная и очно-заочная)); $s_{g_{i_{jk}}}$ – студент $g_{i_{jk}}$-й группы i_{j_k}-й специальности, направления, специализации j-й кафедры k-го факультета (института)
$T_{i_{jk},m_{pd_d}}^{p,t}$	Трудоемкость модуля m_{pd_d} за аттестацию t в зачетных единицах
$T_{i_{jk},m_{pd_d}}^{p}$	Трудоемкость дисциплины (поддисциплины) pd_d по видам занятий за семестр в зачетных единицах
$T_{i_{jk},m_{pd_d}}$	Трудоемкость дисциплины (поддисциплины) pd_d за семестр в зачетных единицах
$T_{i_{jk},pd_d}^{\text{фо}}$	Трудоемкость по форме отчетности (зачет или экзамен) в зачетных единицах по поддисциплине (дисциплине) за семестр
$O_{l,f,s_{g_{i_{jk}}}}^{t,m_{pd_d}}$	Средневзвешенная оценка по модулю
$O_{l,f,s_{g_{i_{jk}}}}^{pd_d}$	Средневзвешенная оценка за семестр
$R_{l,f,s_{g_{i_{jk}}}}^{t,m_{pd_d}}$	Рейтинг $s_{g_{i_{jk}}}$-го студента за аттестацию t по модулю m_{pd_d}
$R_{l,f,s_{g_{i_{jk}}}}^{pd_d}$	Рейтинг $s_{g_{i_{jk}}}$-го студента за семестр по поддисциплине (дисциплине)

Задания

1. Разработать модель вариантов использования.
2. Разработать спецификации вариантов использования.

Контрольные вопросы

1. Какие бывают уровни представления требований?
2. К какому стандарту относится модель вариантов использования?
3. Что такое вариант использования?
4. Какие вы знаете отношения между вариантами использования?
5. Что показывает связь между актантом и вариантом использования?
6. Опишите структуру спецификации варианта использования.
7. Для чего разрабатывается спецификация вариантов использования?

Работа с литературой:

Рекомендуемые источники информации (№ источника)

Основная	Дополнительная	Методическая	Интернет-ресурсы
1-3	1-3	1-3	1-2

Оценочные средства: контрольные вопросы

Тема 3. Анализ требований и их формализация.
Лабораторная работа № 4. Оформление технического задания в соответствии с ГОСТ

Форма проведения: Практическое выполнение задания

Цель работы – сформировать навыки разработки и оформления технического задания на разработку автоматизированных систем в соответствии с ГОСТ 34.602–89.

Краткие теоретические сведения

Техническое задание разрабатывается в соответствии с ГОСТ 34.602–89, основное требование которого заключается в соблюдении структуры ТЗ:

- ТЗ должно состоять из девяти разделов;
- названия разделов должны точно соответствовать требованиям ГОСТа;
- названия подразделов и пунктов могут быть изменены по усмотрению разработчика, но суть разделов должна удовлетворять требованиям ГОСТа.

Ниже даны рекомендации по написанию каждого раздела ТЗ.

Общие сведения

В разделе указывают:

- полное наименование системы и ее условное обозначение;
- шифр темы или шифр (номер) договора;
- наименование предприятий (объединений) разработчика и заказчика (пользователя) системы и их реквизиты;
- перечень документов, на основании которых создается система, с указанием, кем и когда утверждены эти документы;
- плановые сроки начала и окончания работы по созданию системы;
- сведения об источниках и порядке финансирования работ;
- порядок оформления и предъявления заказчику результатов работ по созданию системы (ее частей), изготовлению и наладке отдельных средств (технических, программных, информационных) и программно-технических (программно-методических) комплексов системы.

В данном разделе одним из основных подразделов является «Актуальность разработки системы». Очень важно правильно написать этот подраздел. Для этого нужно четко представлять проблему или проблемы, стоящие перед заказчиком. Представление о проблемах должно сформироваться после проведения системного анализа, описанного выше.

В этом подразделе рекомендуется описывать только те проблемы, которые решаются автоматизацией полностью или частично. То есть внедрение автоматизированной информационной системы, для которой разрабатывается ТЗ, позволит решить полностью или частично описанную здесь проблему.

Проблема должна быть сформулирована ясно и лаконично, в одном или двух предложениях. Затем необходимо раскрыть проблему более подробно и описать пути ее решения. То есть нужно показать, что разрабатываемая система в состоянии решить описанную здесь проблему.

Пример написания раздела

Назначение документа

Техническое задание является основным документом, определяющим общие требования и порядок создания автоматизированной информационной системы (АИС). Включаемые в настоящее ТЗ требования соответствуют современному уровню развития информационных технологий и не уступают аналогичным требованиям, предъявляемым к лучшим отечественным и зарубежным аналогам. Устанавливаемые в настоящем документе требования на АИС не ограничивают разработчика системы в поиске и реализации наиболее эффективных технико-экономических решений.

Все изменения к данному документу оформляются отдельными согласованными документами.

Наименование системы

Полное наименование системы – «Автоматизированная информационная система «Научно-исследовательские работы». Краткое наименование АИС НИР.

Сведения о заказчике и исполнителе

Заказчик системы – «Университет» в лице проректора И. И. Иванова

Исполнитель – ООО «IT-консалтинг» в лице директора П. П. Петрова.

Основания для выполнения работ, сроки и финансирование

Разработка ведется на основании договора № 1 от 01.01.2018, заключенного между «Университетом» и ООО «IT-консалтинг».

Система должна быть разработана в течение 2018 года и сдана в опытную эксплуатацию до 12.12.2018.

Работы по созданию системы финансируются Университетом в соответствии с календарным планом, являющимся неотъемлемой частью договора.

Основные понятия, определения и сокращения

Данный пункт содержит перечень основных понятий, определений и сокращений, используемых в настоящем документе.

Автоматизированная система в защищенном исполнении – автоматизированная система, реализующая информационную технологию выполнения установленных функций в соответствии с требованиями стандартов и/или нормативных документов по защите информации.

Актант (пользователь системы) – субъект (человек, организация, другая АИС), использующий функции или информацию данной системы.

Архитектура системы – высокоуровневая концепция системы и ее окружения.

Архитектура программной системы (в фиксированный момент времени) – организация структуры значимых компонентов системы, взаимодействующих через интерфейсы. Указанные компоненты, в свою очередь, составлены из более мелких компонентов и интерфейсов.

База данных (*БД*) – совместно используемый набор логически связанных данных (и описание этих данных), предназначенных для удовлетворения информационных потребностей организации.

Вариант использования – функциональный связный блок, выраженный в виде транзакции между актантом и системой. Вариант использования описывает поведение системы как последовательности действий. Любой вариант использования должен приводить к полезному результату для актанта.

Доступность информации – состояние информации, характеризуемое способностью АИС обеспечивать беспрепятственный доступ к информации субъектов, имеющих на это полномочия.

Защита информации – деятельность по предотвращению утечки защищаемой информации, несанкционированных и непреднамеренных воздействий на защищаемую информацию.

Конфиденциальная информация – информация с ограниченным доступом, не содержащая сведений, составляющих государственную тайну, доступ к которой ограничивается в соответствии с законодательством Российской Федерации.

Конфиденциальность информации – состояние защищенности информации, характеризуемое способностью АИС обеспечивать сохранение в тайне информации от субъектов, не имеющих полномочий на ознакомление с ней.

Модель – абстрактное представление одного или нескольких аспектов системы. Это полное описание системы с некоторой точки зрения. Одной модели всегда недостаточно для описания всех аспектов системы.

Модель вариантов использования – диаграмма, описывающая основные варианты использования системы, актантов и отображающая связи актантов с вариантами использования (распределение функциональности системы между актантами).

Модуль – элементарный компонент программной системы.

Несанкционированный доступ (*НСД*) – доступ к информации или действия с информацией, нарушающие правила разграничения доступа.

Сервер приложений – специализированное программное обеспечение, предназначенное для централизованного хранения и обработки базы данных.

Система управления базами данных (*СУБД*) – специализированное программное обеспечение, предназначенное для централизованного хранения и обработки данных в БД, а также управления доступом нескольких пользователей к одним и тем же данным.

Спецификация вариантов использования – документ, описывающий основную последовательность взаимодействия актанта с системой (поток) и все альтернативные потоки одного варианта использования.

Утечка информации – неконтролируемое распространение защищаемой информации.

Целостность информации – состояние защищенности информации, характеризуемое способностью АИС обеспечивать сохранность и неизменность информации при попытках несанкционированных или случайных воздействий на нее в процессе обработки или хранения.

Актуальность разработки системы

Актуальность выполнения разработки связана с увеличением объемов и тематики выполняемых в «Университете» исследований, расширением влияния результатов исследований на экономику региона и широким внедрением автоматизированных систем мониторинга и управления научно-инновационным процессом.

Внедрение разработанной АИС НИР позволит упорядочить учет, облегчить поиск информации по проводимым НИР, обеспечить оперативный контроль выполнения календарных и финансовых планов по проектам НИР и по вузу в целом. АИС НИР позволит в дальнейшем подключить к ней филиалы «Университета», расширив сферу ее применения.

Назначение и цели создания (развития) системы

Данный раздел обычно состоит из четырех подразделов:

- цели создания системы;
- назначение системы;
- задачи, решаемые системой;
- область применения системы.

Цель создания системы формулируется как решение проблемы, описанной в подразделе «Актуальность разработки системы».

Назначение системы – это перечень потребностей заинтересованных лиц, которые будут полностью или частично удовлетворены за счет применения разрабатываемой АИС.

В подразделе «Задачи, решаемые системой» описывают те задачи, которые решаются за счет использования системы и в совокупности позволяют достичь заявленной цели и удовлетворить потребности заинтересованных лиц. Задачи, описываемые в данном подразделе, связаны с функциями пользователей системы, которые они выполняют в своей повседневной работе.

В подразделе «Область применения системы» нужно указать границы действия системы. Описывается, кто и в каком объеме будет использовать систему при решении своих задач.

Пример написания раздела

Цели создания системы

Автоматизированная информационная система разрабатывается с целью повышения эффективности управления научным процессом за счет автоматизации процесса регистрации и мониторинга состояния НИР и их динамики.

Назначение системы

АИС НИР предназначена для:

- обеспечения оперативного доступа к информации о научных разработках и ходе выполнения хоздоговоров и грантов;
- формирования всех видов отчетов, связанных с научной деятельностью «Университета»;
- обеспечения руководителей научных проектов инструментом, автоматизирующим большую часть рутинной работы по оформлению результатов НИР и сопутствующих документов.

Задачи, решаемые системой

АИС НИР позволяет решать следующие задачи:

- вести учет НИР;
- выполнять обширный поиск информации по НИР;
- вести календарный и финансовый планы;
- сообщать пользователям системы и предупреждать их о сроках выполнения работ и требуемой отчетности в соответствии с календарным и финансовым планами;
- формировать отчеты по выполненным НИР.

Область применения системы

АИС НИР используется:

- проректором по НИР для мониторинга состояния НИР и эффективности использования финансовых средств;
- начальником Управления НИОКР для мониторинга, оперативного управления организацией НИР и подготовки организационных решений для утверждения проректором по НИР;
- экономической службой УНИОКР для контроля исполнения календарных планов, включая контроль получения финансовых средств и исполнения сметы расходов по каждой НИР;
- руководителями НИР для регистрации НИР, ведения календарных планов и сметы расходов, кадрового состава, основного содержания и результатов завершенных НИР.

Характеристики объекта автоматизации

В настоящем разделе ТЗ приводят:
- краткие сведения об объекте автоматизации или ссылки на документы, содержащие такую информацию;
- сведения об условиях эксплуатации объекта автоматизации и характеристиках окружающей среды;
- количественные характеристики потоков данных и данных, подлежащих хранению.

Для описания объекта автоматизации часто применяют различные методологии, такие как IDEF0, IDEF3, DFD или ARIS. С использованием этих технологий моделируются процессы, которые описываются в тексте. Если объект автоматизации несложный, то он описывается текстом без применения каких-либо технологий.

Пример написания раздела

Организация и планирование научно-исследовательской и инновационной деятельности

«Университет» в целях организации эффективной научной и инновационной деятельности осуществляет:
- организацию экспертизы научных тем и инновационных проектов, представляемых для финансирования в рамках заказа Министерства образования и науки РФ и включения их в число участников научных и научно-технических программ;
- создание центров коллективного пользования уникальным научным оборудованием в кооперации с вузами региона в рамках интегрированных учебно-научно-производственных комплексов, создание системы снабжения оборудованием, средствами вычислительной и оргтехники, материалами и комплектующими изделиями, организацию технического обслуживания и ремонта;
- разработку организационной структуры научно-исследовательской и инновационной деятельности, правовых и экономических основ этого вида работ;
- разработку, внедрение и сертификацию системы менеджмента качества в соответствии с требованиями ИСО 9000–2001 при проведении исследований и разработок, получение лицензий на право проведения специальных НИОКР (научно-исследовательские и опытно-конструкторские разработки);
- контроль за выполнением научных исследований и реализацией инновационных проектов, эффективным использованием и развитием научной и экспериментально-производственной базы вуза.

«Университет» планирует свою научно-инновационную деятельность, финансируемую за счет госбюджета и привлеченных средств, в соответствии с утвержденными в установленном порядке научными и научно-техническими программами и (или) договорами, на основе годовых и перспективных тематических планов.

«Университет» ежегодно разрабатывает сводный (из всех источников финансирования) тематический план НИОКР, который утверждается ректором университета по рекомендации НТС.

Утвержденный годовой тематический план НИОКР «Университета» подлежит обязательному исполнению.

Исполнители научно-исследовательских работ

- Субъектами и инфраструктурными подразделениями научно-исследовательской и инновационной деятельности являются:
- руководящий состав «Университета» (ректор, проректоры);
- институты, факультеты, кафедры, филиалы;
- подразделения (управления, отделы, службы), обеспечивающие непосредственно научно-исследовательскую и инновационную деятельность:

- Управление научно-исследовательских и опытно-конструкторских работ и его отделы;
- Управление стандартизации, сертификации и качества;
- Административно-правовое управление;
- библиотека «Университета»;
- информационно-вычислительный центр;
- Управление бухгалтерского учета и финансового контроля;
- Планово-финансовое управление;
- Управление по режиму и безопасности;
- Хозяйственное управление;
- Управление кадров.

Научные работы в «Университете» выполняются:

- профессорско-преподавательским составом в соответствии с индивидуальными планами (написание монографий, статей, тезисов, заявок на объекты интеллектуальной собственности и др.) в основное рабочее время;
- научными, инженерно-техническими работниками, учебно-вспомогательным персоналом, специалистами и рабочими научных и проектно конструкторских подразделений в основное рабочее время;
- докторантами, аспирантами, стажерами в соответствии с индивидуальными планами их подготовки, а также в свободное от работы (учебы) на кафедрах, в научно-инновационных подразделениях за дополнительную плату;
- студентами, бакалаврами, магистрантами в ходе выполнения курсовых и дипломных проектов, других исследовательских работ, предусмотренных учебными планами, в студенческих научных кружках, студенческих конструкторских бюро, центрах НТТМ, молодежных инновационных центрах, а также на кафедрах, в НИИ и в других структурных подразделениях «Университета» в свободное от учебы время за дополнительную плату либо безвозмездно.

К выполнению научных работ, финансируемых из средств федерального бюджета и по хозяйственным договорам, в том числе на условиях совместительства, привлекаются профессорско-преподавательский состав, научные сотрудники, руководящие и другие работники «Университета», а также сотрудники предприятий, учреждений и организаций, независимо от форм собственности, в свободное от основной работы время по договорам подряда.

Учет и отчетность по научно-исследовательским работам

Все выполняемые в «Университете» НИОКР подлежат государственной регистрации в соответствии с требованиями действующей нормативно-технической документации.

Не подлежат государственной регистрации работы, связанные с обслуживанием научных исследований и предоставлением научно-производственных услуг.

Государственную регистрацию и учет выполняемых и законченных открытых НИР и НИОКР осуществляет Всероссийский научно-технический и информационный центр (ВНТИЦ) Федерального агентства по науке и инновациям.

Подразделение-исполнитель НИОКР через Управление стандартизации, сертификации и качества (УССиК) направляет во ВНТИЦ (Всероссийский научно-технический информационный центр) регистрационную карту (РК) установленного образца не позднее 30 дней с момента начала финансирования работы.

Подразделение-исполнитель зарегистрированной НИОКР через УССиК и УНИОКР в срок, не превышающий 30 дней с момента окончания работы и приемки ее заказчиком, направляет во ВНТИЦ информационную карту (ИК) установленного образца и научно-технический отчет о зарегистрированной НИОКР, утвержденный проректором по НИР.

Представление заключительного отчета о зарегистрированной работе является обязательным при выполнении работы, финансируемой из средств федерального бюджета.

Порядок представления информационных материалов, правила составления регистрационной и информационной карт должны соответствовать Инструкции о порядке регистрации и учета открытых научно-исследовательских и опытно-конструкторских работ, представления по ним отчетов и информационных материалов в ВНТИЦ и выдачи информации этим центром (находится в УССиК).

По завершении НИОКР, финансируемых из внебюджетных источников, в целях рекламы и распространения информации об этих работах во ВНТИЦ вместе с информационной картой представляется рекламно-техническое описание или научно-технический отчет о НИОКР.

Материалы информационной карты используются при формировании электронного банка данных завершенных НИР «Университета» и для размещения информации в сети Интернет.

При представлении отчета о НИОКР университетом (УНИОКР) определяются условия его распространения ВНТИЦ в соответствии с действующим законодательством РФ.

Заключительные отчеты о выполненной работе оформляются в соответствии с требованиями ГОСТ 7.32–2001.

При выполнении опытно-конструкторских и технологических работ заказчику может быть передана рабочая документация, опытные образцы, макеты, технологические регламенты.

Полученные результаты по завершенным этапам и НИОКР в целом, выполняемым за счет средств федерального бюджета, подлежат обсуждению на заседаниях кафедры (отдела, лаборатории) с приглашением представителей головных организаций, представителей других кафедр и структурных подразделений «Университета».

Результаты научно-исследовательской и инновационной деятельности подлежат ежегодному обсуждению на Ученом совете или НТС (научно-технический совет) «Университета».

Приемка госбюджетных НИОКР, финансируемых из средств федерального и местного бюджетов, осуществляется НТС «Университета» после прохождения внутренней экспертизы в порядке, установленном приказом по «Университету».

Отчеты по завершенным госбюджетным НИОКР передаются УНИОКР в библиотеку «Университета» в обязательном порядке, а их электронные версии хранятся в подразделениях-исполнителях и в УНИОКР.

Требования к системе

Данный раздел состоит из следующих подразделов:

- требования к системе в целом;
- требования к функциям (задачам), выполняемым системой;
- требования к видам обеспечения.

В подразделе «Требования к системе в целом» указывают:

- требования к структуре и функционированию системы;
- требования к численности и квалификации персонала системы и режиму его работы;
- показатели назначения;
- требования к надежности;
- требования безопасности;
- требования к эргономике и технической эстетике;
- требования к транспортабельности для подвижных АС;
- требования к эксплуатации, техническому обслуживанию, ремонту и хранению компонентов системы;
- требования к защите информации от несанкционированного доступа;
- требования по сохранности информации при авариях;
- требования к защите от влияния внешних воздействий;
- требования к патентной чистоте;
- требования по стандартизации и унификации;
- дополнительные требования.

Многие пункты данного подраздела можно опускать. Главными являются первые два пункта. В первом из них нужно описать структуру разрабатываемой системы, в которой приводится:

- перечень подсистем, их назначение и основные характеристики, требования к числу уровней иерархии и степени централизации системы;
- требования к способам и средствам связи для информационного обмена между компонентами системы;
- требования к характеристикам взаимосвязей создаваемой системы со смежными системами, требования к ее совместимости, в том числе указания о способах обмена информацией (автоматически, пересылкой документов, по телефону и т. п.);
- требования к режимам функционирования системы;
- требования по диагностированию системы.

Для описания структуры можно приводить схемы и диаграммы с дальнейшим разъяснением по тексту. Для построения схем и диаграмм можно использовать как специализированные средства (например, редакторы UML), так и средства общего назначения (например, Microsoft Office Visio).

В подразделе «Требования к функциям (задачам), выполняемым системой», приводят:

по каждой подсистеме перечень функций, задач или их комплексов (в том числе обеспечивающих взаимодействие частей системы), подлежащих автоматизации;

требования к качеству реализации каждой функции (задачи или комплекса задач), к форме представления выходной информации, характеристики необходимой точности и времени выполнения, требования одновременности выполнения группы функций, достоверности выдачи результатов;

перечень и критерии отказов для каждой функции, по которой задаются требования по надежности.

Для описания функций и требований к качеству их реализации можно использовать метод вариантов использования и их спецификаций, рассмотренных в данных методических указаниях по выполнению лабораторных работ.

Пример написания раздела

Требования к системе в целом

АИС НИР должна быть разработана в виде сайта на основе трехуровневой архитектуры. Сервер баз данных и сервер приложений АИС НИР должны быть созданы на базе постреляционной СУБД Cache. АИС НИР должна иметь два типа клиентских мест: первый предназначен для работы руководителей НИР и должен быть реализован в облегченном варианте на базе технологии CSP; второй тип в связи со сложностью реализуемых функций необходимо разработать на базе Java-технологии (applet – servlet) под платформу J2EE.

АИС обрабатывает конфиденциальную информацию (персональные данные преподавателей, сотрудников, аспирантов и студентов) и представляет собой автоматизированную систему в защищенном исполнении.

Требования к структуре и функционированию системы

АИС НИР должна быть реализована с использованием технологии Cache в виде четырех подсистем. Архитектура системы представлена на рисунке 4.1.

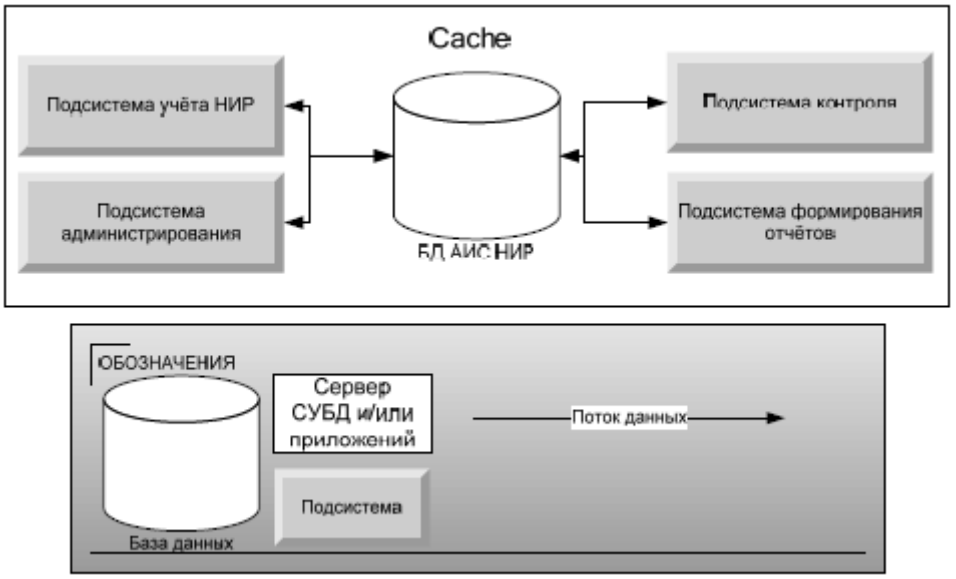


Рисунок 4.1 - Архитектура АИС НИР

Подсистема учета НИР предназначена для осуществления ввода регистрационных данных о сотрудниках и выполняемых НИР, а также информации о результатах выполненных работ. При реализации данной подсистемы необходимо разграничить доступ к данным по НИР в соответствии с установленными уровнями доступа к данным.

Подсистема контроля предназначена для мониторинга выполнения календарного плана по темам, а также выполнения финансового плана поквартально с выдачей информации о невыполнении плана по срокам и объемам по темам.

Подсистема формирования отчетов предназначена для создания отчетов о состоянии квартальных сводок выполнения финансового и календарного планов, а также отчетов о выполнении НИР.

Подсистема администрирования предназначена для регистрации пользователей системы, назначения прав пользователям и редактирования справочных данных.

Требования к численности и квалификации персонала

Пользователями АИС НИР являются:

- профессорско-преподавательский состав в лице руководителей работ по НИР;
- руководящий состав УНИОКР и проректор по НИР.

Пользователи АИС НИР должны:

- иметь навыки работы на ПК в качестве пользователя;
- знать принципы работы с ОС Windows XP/7/8/10;
- пройти обучение для работы с АИС НИР на своем рабочем месте в объеме Руководства пользователя.

Администратор АИС НИР должен иметь высшее образование со специализацией в области разработки информационных систем и баз данных, обладать навыками администрирования современных SQL-серверов и серверов приложений, пройти обучение основам работы с АИС НИР в объеме технической (эксплуатационной) документации (Руководство администратора) на систему.

Требования к функциям (задачам)

В АИС НИР должны быть реализованы функции, представленные ниже на диаграммах вариантов использования. Каждая диаграмма описывает функциональность одной подсистемы (рисунок 4.2).

Рисунок 4.2 - Варианты использования для подсистемы учета НИР

Описание вариантов использования

Редактировать анкету работника по НИР. Анкета заполняется обязательно для руководителя НИР и по желанию для других участников НИР. Форма ввода анкетных данных должна содержать следующие поля: фамилия, имя, отчество; дата рождения; ученая степень; год присуждения ученой степени; ученое звание; год присвоения ученого звания; полное название организации – основного места работы (текстовое поле); сокращенное название организации – основного места работы (текстовое поле); должность; область научных интересов; общее число публикаций; поддержка проектов заявителя в форме грантов; почтовый индекс; почтовый адрес; телефон служебный; телефон домашний; ИНН; номер страхового пенсионного свидетельства; электронный адрес.

Зарегистрировать НИР. Научно-исследовательская работа регистрируется в соответствии с учетной карточкой НИР и регистрационной картой (РК), формы которых приведены ниже.

Для заполнения учетной карточки должны быть реализованы следующие поля:

- наименование НИР;
- регистрационный номер договора (предлагать автонумерацию);
- источник финансирования. Данные выбираются из справочника выпадающим списком (хоздоговор, госбюджет (федеральный, региональный), гранты (ККНФ, РГНФ, РФФИ, международные фонды, другие)), средства спонсоров, централизованный фонд КГПУ, кредиты

банков, другие средства; номер государственной регистрации; характер работы (фундаментальная, поисковая, прикладная, ОКР, разработка);

- сведения о наличии интеллектуальной собственности;
- заказчик (полное наименование и юридический адрес). Заказчик дол-
- жен выбираться из справочника. Если его там нет, то предлагать форму для
- ввода данных в справочник;
- факультет (из справочника);
- кафедра (из справочника);
- научный руководитель (ФИО, ученая степень и звание, стаж работы). Руководитель выбирается из справочника сотрудников вуза;
- кадровый состав НИР. Данные о работниках вводятся из справочника путем прикрепления существующих в справочнике работников к данной НИР. В случае отсутствия работников в справочнике система должна предложить внести работников в справочник;
- организации-соисполнители;
- код ГРНТИ (из справочника).

Ввести результаты работы по НИР. По окончании работы руководителем НИР вводятся результаты в систему. Форма ввода результатов должна содержать следующие поля:

- полученные научные и/или научно-технические результаты (текстовое поле на 2000 знаков);
- полученная научная и/или научно-техническая продукция (текстовое поле на 2000 знаков);
- ключевые слова и словосочетания, характеризующие результаты (продукцию). Текстовое поле на 500 знаков;
- наличие аналогов для сопоставления результатов (продукции) (Да/Нет);
- преимущества полученных результатов (продукции) по сравнению с результатами отечественных и зарубежных НИР (текстовое поле на 2000 знаков);
- степень готовности полученных результатов к практическому использованию (текстовое поле на 500 знаков);
- перспективы, формы и сроки коммерческого использования результатов и продукции (текстовое поле на 2000 знаков);
- форма представления результатов НИР (научно-технические отчеты, учебники, доклады, патенты, экспонаты выставок). Поле заполняется из справочника;
- библиографический список публикаций, отражающих результат работы. Система должна выводить подсказку о правильности заполнения библиографического списка;
- использование результатов в учебном процессе (Да/Нет). Если Да, то вывести текстовое поле для ввода описания на 100 знаков;
- количество сотрудников из числа ППС, принимавших участие в выполнении НИР и указанных научно-технических отчетах в качестве исполнителей;
- количество студентов, принимавших участие в выполнении НИР, в том числе: указанных в научно-технических отчетах в качестве исполнителей; являющихся авторами публикаций по результатам НИР; с оплатой труда за счет средств НИР.

Состав и содержание работ по созданию (развитию) системы

Раздел должен содержать перечень стадий и этапов работ по созданию системы в соответствии с ГОСТ 34.601–90, сроки их выполнения, перечень организаций-исполнителей работ, ссылки на документы, подтверждающие согласие этих организаций на участие в создании системы, или запись, определяющую ответственного (заказчик или разработчик) за проведение этих работ.

В данном разделе также приводят:

- перечень документов по ГОСТ 34.201, предъявляемых по окончании соответствующих стадий и этапов работ;
- вид и порядок проведения экспертизы технической документации (стадия, этап, объем проверяемой документации, организация-эксперт);
- программу работ, направленных на обеспечение требуемого уровня надежности разрабатываемой системы (при необходимости);
- перечень работ по метрологическому обеспечению на всех стадиях создания системы с указанием сроков их выполнения и организаций исполнителей (при необходимости).

Пример написания раздела

Разработка системы должна выполняться на основе архитектурно-ориентированного подхода. Выбранная модель жизненного цикла должна позволять выполнять итеративную и инкрементную разработку системы.

Таблица 4.1 - Перечень работ по созданию АИС

Наименование работы	Результат
Разработка спецификаций вариантов использования (описание последовательностей действий пользователей и системы в рамках каждого варианта использования)	Документы спецификаций
Разработка архитектуры программной системы	Модели архитектуры системы для каждого выбранного архитектурного представления
Уточнение логической структуры АИС (детальное проектирование)	Спецификация логической архитектуры АИС
Разработка модели данных для проектируемой подсистемы или системы в целом и создание БД	Объектная или реляционная модель данных и БД
Разработка проектных моделей пользовательского интерфейса	Модель пользовательского интерфейса модулей АИС в среде разработки
Проектирование, разработка компонентов системы и их тестирование	Действующий образец АИС, функционирующий на программно-аппаратном комплексе разработчика. Сценарии тестов
Интеграционное тестирование функций АИС, исправление кода	Действующий образец АИС, удовлетворяющий требованиям ТЗ
Разработка документации	Комплект пользовательской документации АИС
Установка системы и приемочное тестирование	АИС, соответствующая требованиям ТЗ, установленная на программно-аппаратном комплексе заказчика и готовая к опытной эксплуатации
Обучение пользователей	Пользователи обладают практическими навыками работы с системой
Внедрение в опытную эксплуатацию	Акт сдачи-приемки системы в опытную эксплуатацию
Сопровождение системы (работа по замечаниям пользователей) во время опытной эксплуатации	Список дефектов и предложений по развитию и/или изменению системы

Основной перечень работ по созданию АИС, их содержание и результаты приведены в таблице 4.1. Здесь приведен перечень работ, соответствующий одной итерации жизненного цикла. Предполагается, что все перечисленные работы будут повторяться на каждой итерации при реализации подсистемы или отдельных вариантов использования.

Порядок контроля и приемки системы

В данном разделе указывают:

виды, состав, объем и методы испытаний системы и ее составных частей (виды испытаний в соответствии с действующими нормами, распространяющимися на разрабатываемую систему);

общие требования к приемке работ по стадиям (перечень участвующих предприятий и организаций, место и сроки проведения), порядок согласования и утверждения приемочной документации;

статус приемочной комиссии (государственная, межведомственная, ведомственная).

Пример написания раздела

Для взаимодействия Исполнителя и Заказчика в организации Заказчика определяется эксплуатационная служба и назначается сотрудник, ответственный за приемку системы.

Разработанная система принимается в опытную эксплуатацию. Готовые компоненты системы могут передаваться поочередно. Сдача и приемка автоматизированной информационной системы осуществляется на основе результатов тестирования, проводимого представителями Заказчика и Исполнителя в соответствии с программой испытания, которая формируется

совместно. В программе испытания должны быть указаны виды, состав, объем и методы проверки правильности получения выходных данных и соответствия системы требованиям данного ТЗ.

Для проверки работоспособности системы проводится выполнение контрольных примеров. Составление контрольных примеров с последующей их передачей комиссии производится эксплуатационной службой и разработчиками совместно. Для выполнения контрольного примера должен быть предоставлен программно-аппаратный комплекс, удовлетворяющий требованиям, изложенным в подразделе «Требования к видам обеспечения» настоящего документа. По результатам выполнения тестов комиссией составляется перечень замечаний, который рассматривается разработчиком в течение трех дней.

Опытная эксплуатация призвана выявить ошибки и собрать замечания и проводится в обязательном порядке. Для обеспечения проведения опытной эксплуатации формируется комиссия по приемке системы, в состав которой входят эксплуатационная служба и разработчики.

По окончании опытной эксплуатации эксплуатационная служба передает в комиссию по приемке системы перечень замечаний по работе системы. Комиссия рассматривает замечания и принимает решение о готовности системы к промышленной эксплуатации. В случае подтверждения комиссией готовности системы к промышленной эксплуатации в течение семи дней подписывается акт сдачи-приемки системы в промышленную эксплуатацию.

В противном случае комиссия передает разработчикам согласованный протокол замечаний. После устранения замечаний проводится повторная опытная эксплуатация на усеченном временном интервале.

Система считается сданной в промышленную эксплуатацию после подписания акта сдачи-приемки системы в промышленную эксплуатацию должностным лицом, ответственным за приемку системы. При выявлении существенных несоответствий характеристик системы требованиям ТЗ Заказчиком составляется обоснованный перечень замечаний, который подписывается ответственным лицом Заказчика и передается разработчикам для доработки системы.

Требования к составу и содержанию работ по подготовке объекта автоматизации к вводу системы в действие

В настоящем разделе необходимо привести перечень основных мероприятий (и их исполнителей), которые следует выполнить при подготовке объекта автоматизации к вводу АИС в действие.

В перечень основных мероприятий включают:

- приведение поступающей в систему информации (в соответствии с требованиями к информационному обеспечению) к виду, пригодному для ввода и обработки в АИС;
- изменения, которые необходимо осуществить в объекте автоматизации;
- создание условий функционирования объекта автоматизации, при которых гарантируется соответствие создаваемой системы требованиям, содержащимся в ТЗ;
- создание необходимых для функционирования системы подразделений и служб;
- сроки и порядок комплектования штатов и обучения персонала.

Пример написания раздела

Для подготовки АИС к вводу в эксплуатацию необходимо:

- назначить должностное лицо в организации Заказчика, ответственное за приемку системы;
- установить комплекс технических средств, удовлетворяющих требованиям соответствующего ТЗ, на рабочие места сотрудников организации Заказчика, которые должны участвовать в эксплуатации АИС;
- совместно с Исполнителем выполнить инсталляцию системного ПО в соответствии с Руководством администратора;
- провести ввод данных справочной информации и настройку системы в соответствии с Руководством администратора;
- совместно с Исполнителем составить документ «Программа испытаний»;
- провести испытания в соответствии с документом «Программа испытаний»;
- при удовлетворительном результате испытаний подписать акт технической готовности системы к опытной эксплуатации. При наличии замечаний составить документ «Перечень предложений и замечаний для доработки системы»;
- при необходимости провести обучение потенциальных пользователей АИС основам компьютерной грамотности;

- провести обучение потенциальных пользователей работе с АИС в объеме Руководства пользователя.

Для обеспечения функционирования системы необходимо разработать регламент эксплуатации, предусматривающий работу пользователей и служб сопровождения.

Создание служб, необходимых для функционирования системы

Функционирование АИС должна обеспечивать эксплуатационная служба – структурное подразделение или системный администратор, отвечающие за поддержку работы системы и контроль выполнения требований, изложенных в настоящем документе.

В целях планирования развития системы данная служба должна собирать заявки пользователей, подписанные руководителем соответствующих организационных подразделений, обобщать их и передавать разработчику системы. Для решения этих задач служба сопровождения должна выполнять следующие функции:

- проводить диагностику АИС;
- своевременно проводить резервное копирование баз;
- при возникновении аварийных ситуаций ликвидировать их последствия и восстанавливать технологический режим функционирования АИС;
- регистрировать ошибки, выявленные пользователями в процессе работы с системой, и оперативно передавать их разработчику системы;
- выполнять требования к эксплуатации и техническому обслуживанию АИС;
- проводить настройку автоматизированных рабочих мест пользователей в соответствии с их должностными обязанностями.

Для качественного выполнения перечисленных выше функций все сотрудники рассматриваемого подразделения должны пройти обучение и быть аттестованы разработчиком АИС. Сотрудники, не прошедшие аттестацию, не должны допускаться к выполнению администрирующих функций АИС.

Функциональные этапы внедрения системы

Внедрение системы осуществляется по мере реализации отдельных подсистем или функций системы. В таблице 4.2 представлен порядок внедрения первоочередных функций системы.

Таблица 4.2 Порядок внедрения первоочередных функций системы

№ п/п	Действия с системой и функции системы, подлежащие тестированию	Объем контрольных испытаний	Исполнители
1	Установка базы данных и сервера приложений	Первоначальная и обновления по мере модификаций	ИВЦ
2	Установка клиентских мест	Не менее 60 мест	ИВЦ
3	Генерация ведомостей	Ведомости по всем аттестациям для студентов первого курса	ИВЦ
4	Внесение данных о текущей трудоемкости модулей, внесение результатов по всем видам аттестаций	Все оценки для всех аттестаций студентов первого курса	Кафедры, ведущие занятия на первом курсе
5	Управление статусом ведомостей	Все ведомости должны пройти жизненный цикл от «новых» до «утвержденных»	Кафедры, ведущие занятия на первом курсе
6	Перевод результатов аттестации за семестр в пятибалльную систему и экспорт данных в АИС «Сессия»	В полном объеме для формирования протоколов стипендиальных комиссий для студентов первого курса.	ИВЦ
7	Функции обмена информацией	–	Все участники

Требования к документированию

В разделе приводят:

- согласованный разработчиком и заказчиком системы перечень подлежащих разработке комплектов и видов документов, соответствующих требованиям ГОСТ 34.201 и нормативной технической документации отрасли заказчика; перечень документов, выпускаемых на машинных носителях; требования к микрофильмированию документации;
- требования по документированию комплектующих элементов межотраслевого применения в соответствии с требованиями ЕСКД и ЕСПД;
- при отсутствии государственных стандартов, определяющих требования к документированию элементов системы, дополнительно включают требования к составу и содержанию таких документов.

Пример написания раздела

Комплект сопровождающей документации должен состоять из следующих документов:

- Паспорт системы;
- Общее описание системы;
- Руководство пользователя;
- Руководство администратора;
- Руководство программиста;
- Регламент эксплуатации.

Паспорт системы

Документ «Паспорт системы» должен описывать состав и краткое назначение основных элементов системы, передаваемой Заказчику, и включать следующие разделы:

- *общие сведения*:
  - наименование АИС, ее обозначение, присвоенное разработчиком;
  - наименование организации-разработчика;
- *основные характеристики АИС*:
  - состав функций, реализуемых АИС;
  - описание принципа функционирования;
  - общий регламент и режимы функционирования;
  - сведения о совместимости с другими системами;

- *комплектность* – перечень всех непосредственно входящих в состав АИС комплексов программных средств, в том числе носителей данных и эксплуатационных документов.

Общее описание системы

Документ «Общее описание системы» должен содержать следующие разделы:

- *назначение системы*:
  - вид деятельности, для информатизации которой предназначена система;
  - перечень объектов автоматизации, на которых будет использоваться система;
- *описание системы*:
  - структура системы и назначение ее частей;
  - сведения о АИС в целом и его составных частях;
  - описание функционирования системы;
  - описание взаимосвязи АИС с другими системами;
  - перечень функций, реализуемых системой.

Руководство администратора

Документ «Руководство администратора» должен содержать всю необходимую информацию, достаточную для работы системного администратора с данной АИС:

- функции администрирования при применении данной АИС;
- процедуры по инсталляции и подготовке АИС к эксплуатации;
- инструкции по тестированию и описание тестового примера;
- инструкции по сохранению и восстановлению данных;
- техническое описание структуры системы и модели данных.

Руководство пользователя

Документ «Руководство пользователя» должен содержать описание пользовательского интерфейса и действий пользователя, достаточное для работы специально обученного пользователя. Данный документ должен содержать следующие разделы:

- введение;
- область применения;
- краткое описание возможностей;
- требования к уровню подготовки пользователя;
- перечень эксплуатационной документации, с которой необходимо ознакомиться пользователю;
- описание пользовательского интерфейса;
- описание процесса импорта данных из смежных систем.

Регламент эксплуатации

Документ «Регламент эксплуатации» должен содержать всю необходимую информацию об использовании системы в работе отделов и отдельных сотрудников в рамках их основной деятельности. В документе должны быть отражены все процессы деятельности отделов, в которых используется АИС, и описан порядок действий сотрудников с использованием АИС.

При проведении сертификации техническая документация должна отвечать действующим государственным стандартам (ГОСТ 34.602–89, ГОСТ 19.201–78, ГОСТ 19.202–78, ГОСТ 19.402–78, ГОСТ 19.502–78, ГОСТ 19.504–79, РД 50-34.698–90, ГОСТ 19.301–79).

Источники разработки

В данном разделе должны быть перечислены документы и информационные материалы (технико-экономическое обоснование, отчеты о завершенных научно-исследовательских работах, информационные материалы на отечественные, зарубежные системы-аналоги и др.), на основании которых разрабатывалось ТЗ и которые должны быть использованы при создании системы.

Правила оформления ТЗ

Разделы и подразделы ТЗ должны быть размещены в порядке, установленном в ГОСТ 34.602–89.

Номера листов (страниц) проставляют начиная с первого листа, следующего за титульным листом, в верхней части листа (над текстом, посередине) после обозначения кода ТЗ.

На титульном листе помещают подписи Заказчика, Разработчика и представителей согласующих организаций, которые скрепляют гербовой печатью. При необходимости титульный лист оформляют на нескольких страницах. Подписи разработчиков ТЗ и должностных лиц, участвующих в согласовании и рассмотрении проекта ТЗ, помещают на последнем листе.

При необходимости на титульном листе ТЗ допускается помещать установленные в отрасли коды, например: гриф секретности, код работы, регистрационный номер ТЗ и др.

Титульный лист дополнения к ТЗ оформляют аналогично титульному листу ТЗ. Вместо наименования «Техническое задание» пишут «Дополнение № ... к ТЗ на АИС ... ».

На последующих листах дополнения к ТЗ помещают основание для изменения, содержание изменения и ссылки на документы, в соответствии с которыми вносятся эти изменения.

При изложении текста дополнения к ТЗ следует указывать номера соответствующих пунктов, подпунктов, таблиц основного ТЗ и т. п. и применять слова «заменить», «дополнить», «исключить», «изложить в новой редакции».

Задание

Написать разделы ТЗ и оформить его в соответствии с ГОСТ 34.602–89.

Контрольные вопросы

1. Структура технического задания по ГОСТу.
2. Какие допущения регламентирует ГОСТ при написании ТЗ?
3. В каких разделах ТЗ используется материал предыдущих лабораторных работ?
4. Какими ГОСТами и руководящими документами нужно руководствоваться при написании раздела «Требования к документированию»?
5. Какой ГОСТ регламентирует оформление ТЗ?__

Работа с литературой:

Рекомендуемые источники информации (№ источника)			
Основная	Дополнительная	Методическая	Интернет-ресурсы
1-3	1-3	1-3	1-2

Оценочные средства: контрольные вопросы

Тема 4. Архитектуры программных систем.

Лабораторная работа № 5. Реализация архитектуры на базе объектно-реляционного отображения с типизированными объектами

Форма проведения: Практическое выполнение задания

Цель работы – сформировать навыки:

- проектирования архитектуры клиент-серверных приложений;
- разработки системы на примере трехслойной архитектуры с использованием объектно-реляционного отображения;
- работы в Borland Developer Studio.

Краткие теоретические сведения

Трехслойная архитектура на базе объектно-реляционного отображения с типизированными объектами

Трехслойная архитектура реализуется на стороне клиента и имеет следующие слои: графический пользовательский интерфейс, бизнес-логику, объектно-реляционное отображение (рисунок 5.1). Взаимодействие слоев осуществляется посредством интерфейса и только в одностороннем порядке, т. е. классы верхнего слоя могут посылать сообщения классам нижнего слоя, но не наоборот.

Графический пользовательский интерфейс

Графический пользовательский интерфейс содержит классы интерфейсных объектов и классы управления интерфейсом пользователя. В слое бизнес-логики содержатся классы, управляющие логикой приложения и реализующие всю функциональность, связанную с обработкой данных и вычислениями. В слое объектно-реляционного отображения находятся классы, выполняющие функции преобразования данных из реляционного вида в объектный и наоборот, а также функции по работе с реляционной базой данных.

Рисунок 5.1 - Трехслойная архитектура

Понятие «типизированные объекты» связано со слоем графического интерфейса. Классы этого слоя напрямую могут работать с объектами предметной области, ссылки на которые они получают от классов бизнес-логики.

В интерфейсе таким образом может выполняться привязка компонентов формы к атрибутам класса предметной области.

Бизнес-логика. Данный слой должен содержать классы, реализующие функции обработки данных и вычислений.

Приведем пример базового (основного) класса, который может создаваться в этом слое.

Таблица 5.1 - Поля класса TDataPrepare

Поле	Описание
fExecute: TExecuteObject	Объект, используемый для выполнения запросов, не возвращающих набор данных
fSelect: TExecuteObject	Объект, используемый для выполнения запросов, возвращающих набор данных

Таблица 5.2 - Свойства класса TDataPrepare

Свойство	Описание
DataSource: TDataSource	Источник данных, используемый для отображения списка объектов
id: integer	Идентификатор объекта (совпадает с Id записи в таблице базы данных). Используется для хранения идентификатора текущего объекта

Таблица 5.3 - Методы класса TdataPrepare

Метод	Описание
Add()	Добавляет объект в БД
Update()	Изменяет атрибуты объекта в БД
Delete()	Удаляет объект из БД
Select()	Выбирает список объектов для отображения
LoadCurrent()	Загружает в поля класса атрибуты текущего объекта
Refresh()	Обновляет список объектов
Create(Connection: TConnection; Transaction: TTransactionObject=nil)	Конструктор класса

Назовем этот класс TDataPrepare. Он должен быть порожден от классаи TDBObject, поскольку должен уметь работать с соединениями и транзакциями. Атрибуты класса представлены в таблице 5.1, свойства класса – в таблице 5.2, виртуальные методы класса – в таблице 5.3.

Объектно-реляционное отображение. Данный слой состоит из классов, которые с помощью драйверов доступа к БД осуществляют взаимодействие с СУБД.

На рисунке 5.2 приведен пример базовых классов, на основе которых можно строить слой объектно-реляционного отображения. В таблице 5.4 представлено описание классов, изображенных на рисунке 5.2.

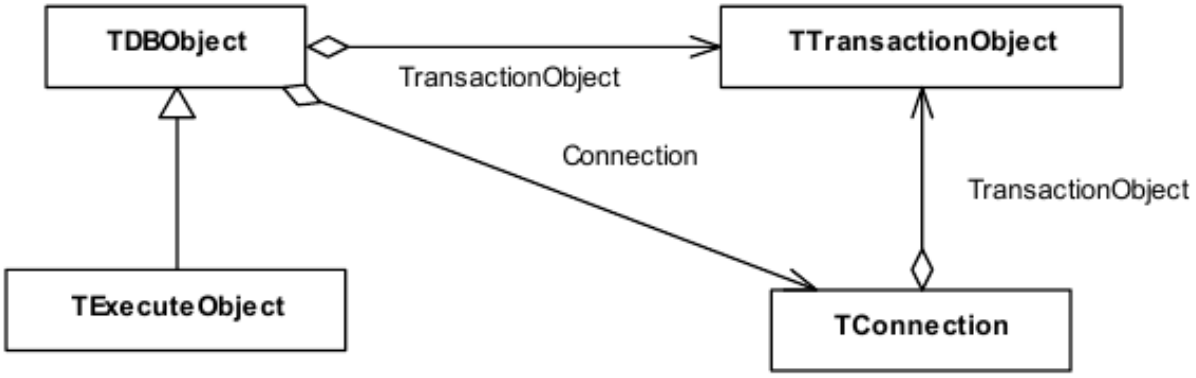


Рисунок 5.2 - Классы объектно-реляционного отображения

Таблица 5.4 - Описание базовых классов объектно-реляционного отображения

Имя класса	Описание
TConnection	Предназначен для управления соединением к базам данных и настройки параметров соединения
TTransactionObject	Предназначен для управления транзакцией (открытие, закрытие, откат)
TDBObject	Предназначен для работы с базой данных. Использует классы TConnection и TTransactionObject
TExecuteObject	Обеспечивает работу с SQL

Характеристики класса TConnection. Свойства класса TConnection представлены в таблице 5.5, методы этого класса – в таблице 5.6.

Таблица 5.5 - Свойства класса TConnection

Свойство	Описание
ServerName: string	Сетевое имя сервера СУБД
DataBaseName: string	Полный локальный путь к файлу БД
UserName: string	Имя пользователя СУБД, используемое для авторизации
Password: string	Пароль доступа к СУБД
DataBase: TIBDatabase	Компонент доступа к базе данных, с которой работает объект
TransactionObject: TTransactionObject	Транзакция, с которой работает объект

Таблица 5.6 - Методы класса TConnection

Метод	Описание
Connect()	Открытие соединения с базой данных
Disconnect()	Закрытие соединения с базой данных

Характеристики класса TTransaction Object.

Класс TTransactionObject является оболочкой к компоненту TIBTransaction, реализующему функции по работе с транзакциями.

Свойства класса TTransactionObject представлены в таблице 5.7, методы данного класса приведены в таблице 5.8.

Таблица 5.7 - Свойства класса TTransactionObject

Свойство	Описание
Transaction: TIBTransaction	Определяет, с какой транзакцией работает объект
InTransaction: Boolean	Определяет, запущена ли транзакция у объекта

Таблица 5.8 - Методы класса TTransactionObject

Метод	Описание
StartTransaction()	Запуск транзакции
Commit()	Подтверждение транзакции
Rollback()	Отмена транзакции

Характеристики класса TDBObject. Поля класса TDBObject представлены в таблице 5.9, свойства данного класса – в таблице 5.10.

Класс TDBObject содержит те же методы, что и класс TTransactionObject, для реализации возможности работы со своей собственной транзакцией. Соответственно если он использует чужую транзакцию, он ничего не может с ней делать. Методы класса TDBObject представлены в табл. 5.11.

Таблица 5.9 - Поле класса TDBObject

Поле	Описание
fOwn_transaction: Boolean	Определяет, использует объект свою транзакцию или чужую, переданную при создании класса

Таблица 5.10 - Свойства класса TDBObject

Свойство	Описание
Connection: TConnection	Подключение, используемое данным объектом
TransactionObject: TTransactionObject	Транзакция, используемая данным объектом

Таблица 5.11 - Методы класса TDBObject

Метод	Описание
StartTransaction()	Запуск транзакции
Commit()	Подтверждение транзакции
Rollback()	Отмена транзакции

Свойства класса TExecuteObject представлены в таблице 5.12, методы данного класса приведены в таблице 5.13.

Таблица 5.12 - Свойства класса TExecuteObject

Свойство	Описание
nfi[name: string]: Integer	Свойство доступа к полям набора данных. Используется, когда поле является целочисленным
nfs[name: string]: string	Свойство доступа к полям набора данных. Используется, когда поле является строковым
nff[name: string]: Extended	Свойство доступа к полям набора данных. Используется, когда поле является дробным

Таблица 5.13 - Методы класса TExecuteObject

Метод	Описание
ExecuteQuery(sql: string; params: array of variant)	Выполняет запрос к базе данных и не возвращает набор данных
SelectQuery(sql: string; params: array of variant; FetchAll: boolean=false)	Выполняет запрос к базе данных и возвращает набор данных
SelectStrings(sql: string; params: array of variant): TStringList	Выполняет запрос к базе данных и возвращает коллекцию строк в классе TStringList из первого поля запроса

Структура БД. В рассматриваемом в лабораторной работе примере будет использоваться база данных, реляционная модель которой приведена на рисунке 5.3. Описание таблиц базы данных приведено в таблице 5.14.

Таблица 5.14 - Описание таблиц базы данных

Таблица	Описание
Books	Хранит информацию о книгах
Groups	Хранит информацию о группах, в которых учатся студенты
Students	Хранит информацию о студентах
Students_Book	Хранит информацию о выданных студенту книгах

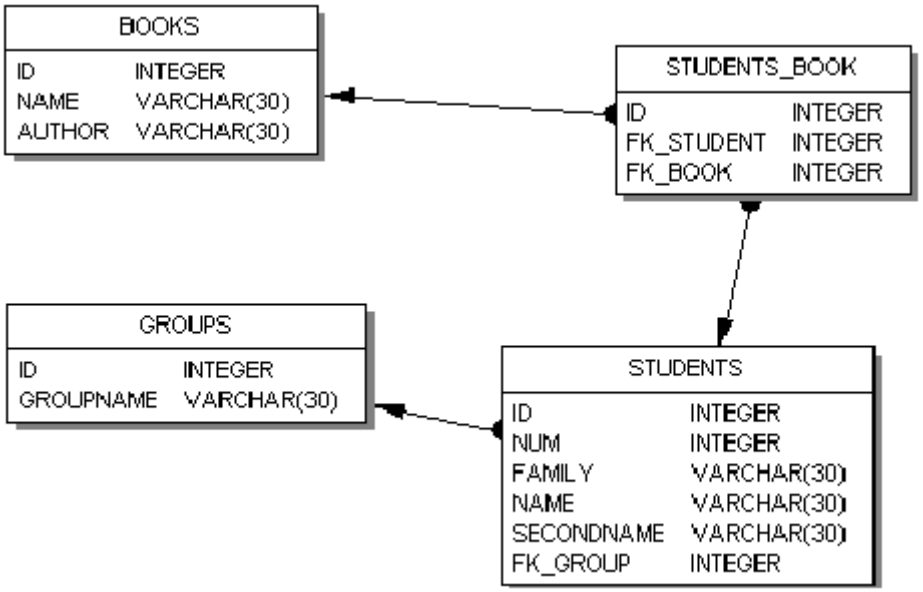


Рисунок 5.3 - Структура базы данных выдачи книг в библиотеке

Создание проекта в Borland Developer Studio

Единицей разработки в Borland Developer Studio 2006 является проект, представляющий файл, в котором хранятся ссылки на все формы и модули.

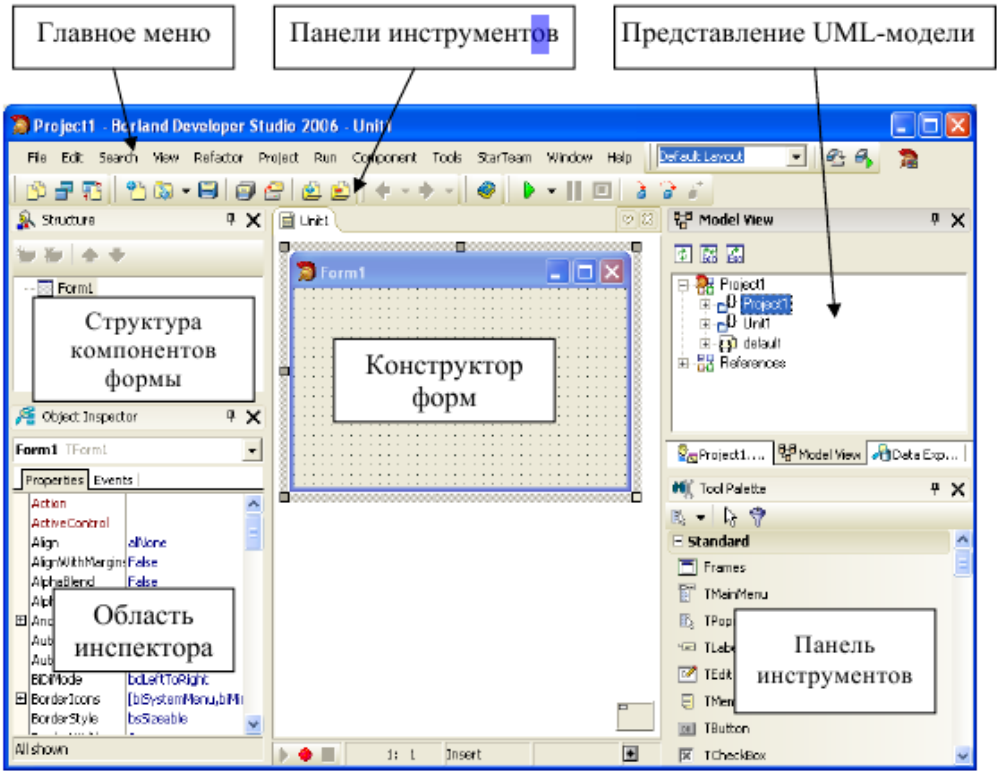


Рисунок 5.4 - Внешний вид Borland Developer Studi

Чтобы создать проект, необходимо выбрать в главном меню пункт File → New → VCL Forms Applications – Delphi. При этом будет создан проект с простым интерфейсом. Вид окна Borland Developer Studio представлен на рисунке 5.4.

Добавление нового модуля в проект

Модули представляют собой структурные единицы, из которых состоит проект. В одном модуле может содержаться несколько классов с описанием их интерфейсов и реализаций.

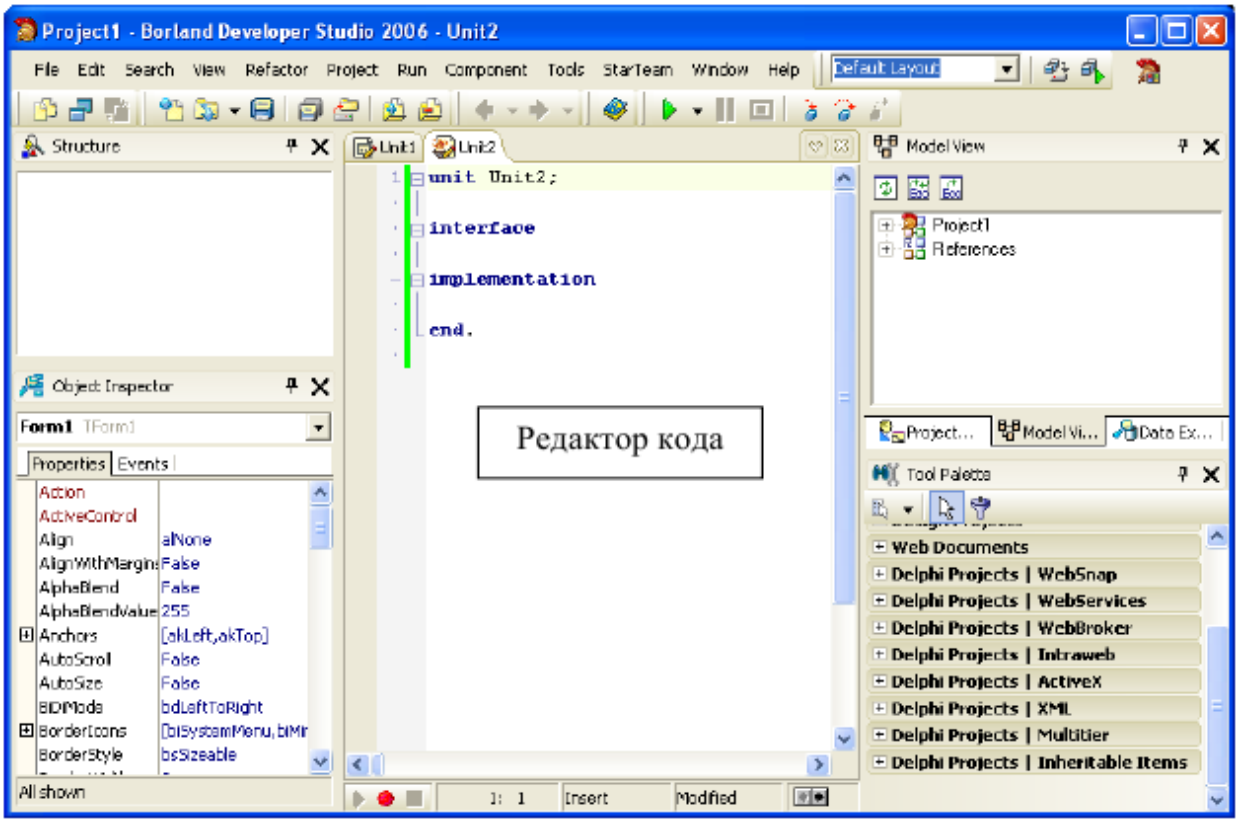


Рисунок 5.5 - Новый модуль

Добавление нового модуля осуществляется путем выбора в главном меню пункта File → New → Unit – Delphi. В правой части редактора в Дереве проекта появится новый элемент (Unit). Модификация кода выполняется в редакторе кода для выбранного модуля (рисунок 5.5).

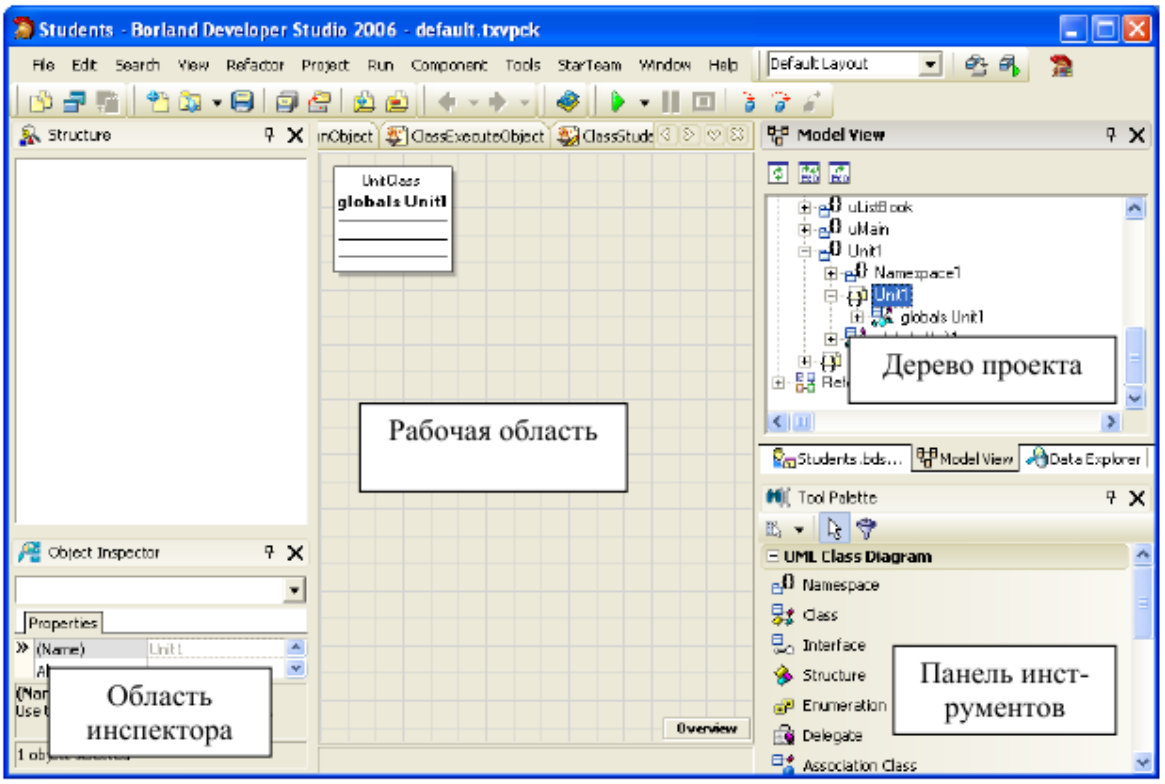


Рисунок 5.6 - Представление UML

Создание классов с помощью диаграммы UML

Для создания класса с помощью диаграммы UML переключитесь на закладку Model View и дважды щелкните по модулю, в который собираетесь добавить класс или несколько классов (рисунок 5.6). Далее на панели инструментов выберите Class и щелкните в Рабочей области. Так же на Рабочую область можно переносить классы, расположенные в других модулях проекта. Для этого необходимо выбрать класс в Дереве проекта и перетащить его на Рабочую область. Перенос других классов на Рабочую область позволяет использовать наследование и агрегацию классов.

Добавление полей. Для добавления поля в класс необходимо щелкнуть правой кноПКй мыши по классу. Появится контекстное меню, в котором нужно выбрать пункт Add и подпункт Field или нажать сочетание клавиш Ctrl + W. При этом в Рабочей области у класса, для которого вы добавили поле, появится новая строка, в которой указывается название поля и его тип. По-другому вы можете настроить свойства поля в Области инспектора.

Добавление свойств. Добавить свойство можно посредством контекстного меню и выбора пункта Add. Далее выберите подпункт Property.

Добавление процедуры. Для добавления процедуры щелкните правой кноПКй мыши по классу и в появившемся контекстном меню выберите пункт Add и подпункт Procedure. Для перехода к определению процедуры нужно выбрать элемент и вызвать контекстное меню, в котором выбирается пункт Go to definition.

В таблице 5.15 представлены свойства, которые можно настроить в Области инспектора для процедуры.

Таблица 5.15 - Свойства процедуры

Свойство	Описание
Override	Определяет, является ли эта процедура переопределенной
Params	Определяет список формальных параметров процедуры
Virtual	Определяет, является ли процедура виртуальной
Visibility	Определяет директиву видимости процедуры

Добавление функции. Для добавления функции в класс щелкните правой кноПКй мыши по классу. Появится контекстное меню, в котором нужно выбрать пункт Add и подпункт Function или нажать сочетание клавиш Ctrl + M. Свойства функции, которые можно настроить в Области инспектора представлены в таблице 5.16.

Таблица 5.16 - Свойства функции

Свойство	Описание
Override	Определяет, является ли эта функция переопределенной
Params	Определяет список формальных параметров функции
Returns	Определяет тип возвращаемого значения функции
Virtual	Определяет, является ли функция виртуальной
Visibility	Определяет директиву видимости функции

Создание отношений между классами

Ассоциация. Между двумя классами устанавливается отношение ассоциации, если между этими классами есть связь. У ассоциации указываются роли и мощности.

Для установки ассоциации между двумя классами поместите на диаграмму эти классы из Дерева проекта, выберите на панели инструментов пункт Association, щелкните мышкой по одному, а затем по второму классу.

В свойствах ассоциации по умолчанию отсутствует направленность, что подразумевает двунаправленную связь между классами. Для установки однонаправленной ассоциации нужно установить свойство Directed в положение true. Мощность и роль устанавливаются в двунаправленной ассоциации на обоих полюсах, в однонаправленной ассоциации – на одном полюсе по направлению стрелки. Для установки мощности выберите из списка значение для свойства Client Cardinality и/или для свойства Supplier Cardinality.

Для установления ролей заполните свойства Client Role и/или Supplier Role соответственно.

Агрегация. Между двумя классами устанавливается отношение агрегации, если один объект одного класса включает в себя объекты другого класса. Для установки типа ассоциации «агрегация» укажите в свойстве ассоциации Type значение Aggregation.

Наследование. Между классами устанавливается отношение наследования, если один класс является базовым (родитель), а второй – дочерним (потомком). При этом дочерний класс наследует свойства и методы родительского класса и реализует свои собственные.

Для установки отношения наследования между классами выберите на панели инструментов компонент Generalization и установите эту связь между классами.

Пример создания классов

Создание классов и отношений между ними слоя объектно-реляционного отображения

В качестве примера создания отношений между классами возьмем

классы слоя объектно-реляционного отображения (рисунок 5.2).

Для создания этих классов необходимо выполнить следующие действия:

1. Создать новый модуль. Для этого в главном меню выберите File → New → Unit – Delphi.

2. Сохраните его, выбрав в главном меню File → Save или нажав сочетание клавиш Ctrl + S, указав при этом имя модуля ClassConnection.

3. Выберите в верхнем правом окне вкладку Model View. Найдите там только что созданный модуль и два раза щелкните по нему левой кноПКй мышки. Окно Borland Studio будет иметь вид, представленный на рисунке 5.6.

4. Перетащите в Рабочую область пункт Class из панели инструментов.

5. Переименуйте только что созданный класс. Для этого щелкните один раз по заголовку класса и введите новое название, либо выберите класс на Рабочей области и настройте свойства Name в Области инспектора. Назовем только что созданный класс TConnection.

6. Для добавления свойства нужно щелкнуть по классу правой кноПКй мыши. При этом появится контекстное меню, в нем выберите пункт Add, а затем пункт Property – появится новое поле. Для его переименования нужно выбрать появившееся свойство и щелкнуть по нему один раз – появится поле ввода, в котором можно указать название и тип свойства. По умолчанию поля

создаются с доступом к записи и чтению через процедуры. Если это не нужно, можно вручную отредактировать свойства в редакторе кода. Создадим свойства согласно таблице 5.5.

7. Добавьте процедуры в соответствии с таблицей 5.6. Для добавления процедуры щелкните правой кноПКй мыши по классу и в появившемся контекстном меню выберите пункт Add и подпункт Procedure.

8. Для добавления функции в классе щелкните правой кноПКй мыши по нему. Появится контекстное меню, в котором нужно выбрать пункт Add и подпункт Function или нажать сочетание клавиш Ctrl + M.

Остальные классы создаются аналогично. Далее установим наследование между классами TExecuteObject и TDBObject.

1. Для установления отношения наследования у класса TExecuteObject нужно найти класс TDBObject во вкладке Model View и перенести на диаграмму класса TExecuteObject. На диаграмме появится ссылка на класс.

2. Выберите на панели инструментов пункт Generalization.

3. Щелкните мышкой по классу TExecuteObect.

4. Щелкните мышкой по классу TDBObject.

Далее установим ассоциацию между классами TDBObject и TTransactionObject.

1. Для настройки ассоциации между классами TDBObject и TTransactionObject нужно перенести класс TDBObject на диаграмму класса TTransactionObject.

2. Выберите на панели инструментов пункт Association.

3. Щелкните по классу TDBObject.

4. Щелкните по классу TTransactionObject.

5. В Области инспектора установите у свойства Type значение Aggregation.

Создание классов слоя бизнес-логики

Структура классов бизнес-логики представлена на рисунок 5.7.

Класс TDataPrepare является базовым классом слоя бизнес-логики. Поля класса представлены в таблице 5.1, свойства класса – в таблице 5.2. Данный класс имеет виртуальные методы, приведенные в таблице 5.3.

Классы, порожденные от класса TDataPrepare. В этих классах необходимо переопределить все виртуальные методы класса TDataPrepare и создать свойства согласно таблицам 5.17 - 5.19.

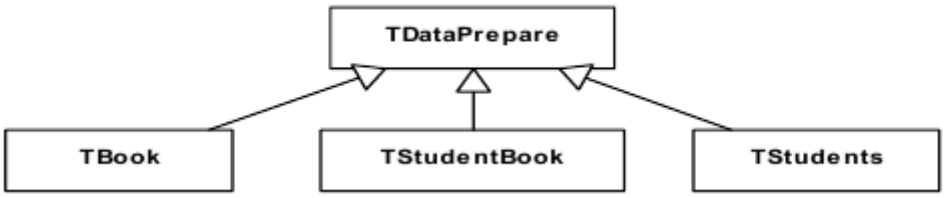


Рис. 5.7. Структура классов слоя «Бизнес-логика»

Таблица 5.17 - Свойства класса TBook

Свойство	Описание
Author: string	Автор книги
Name: string	Название книги

Таблица 5.18 - Свойства класса TStudentBook

Свойство	Описание
Fk_book: integer	Ссылка на книгу, выданную студенту
Fk_student: integer	Ссылка на студента, которому выдали книгу

Таблица 5.19 - Свойства класса TStudents

Свойство	Описание
Family: string	Фамилия студента
Name: string	Имя студента
SecondName: string	Отчество студента
Num: integer	Номер зачетной книжки студента
Group: string	Группа студента

Невизуальные компоненты интерфейса, используемые в примере

TimageList предназначен для хранения и последующего использования различных изображений.

Для использования этого компонента поместите его на форму и двойным щелчком мыши запустите Мастер создания изображений (рисунок 5.8).

В Мастере создания изображений можно добавлять, редактировать удалять изображения формата иконок.

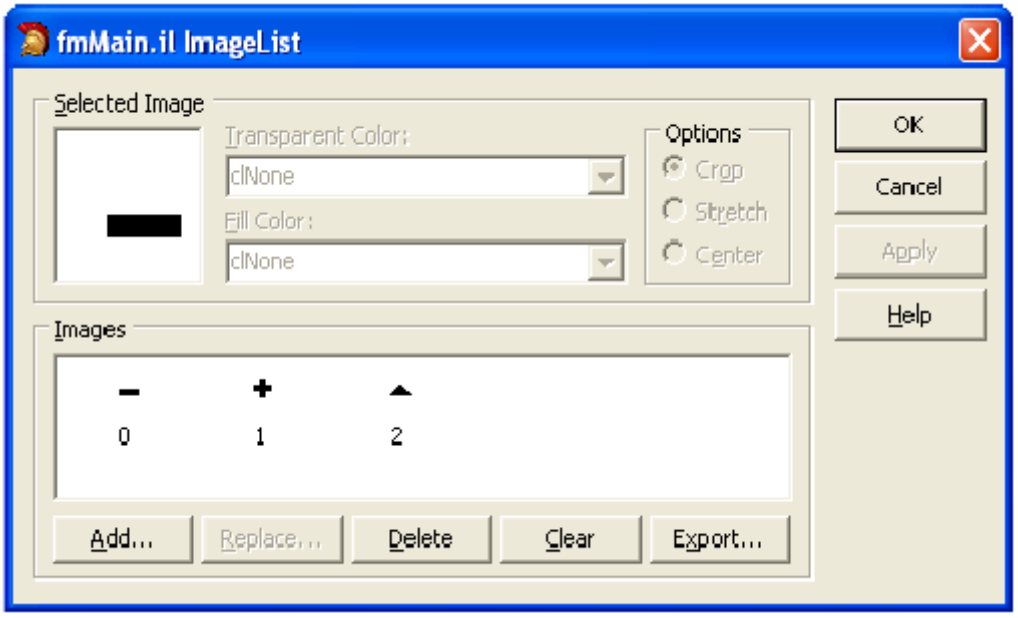


Рисунок 5.8 - Мастер создания изображений в TImageList

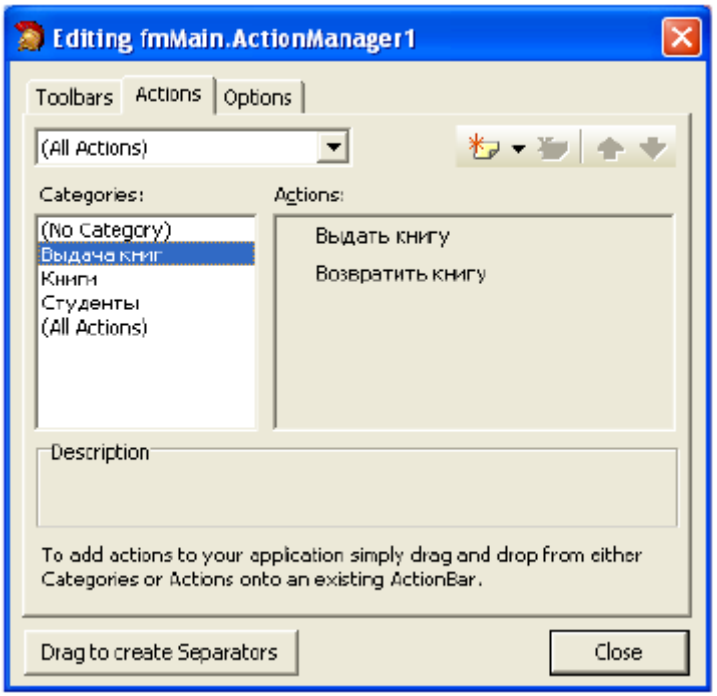


Рисунок 5.9 - Окно настройки действий компонента TActionManager

TActionManager – менеджер действий. Применяется для управления действиями, повторно используемыми в проекте. Для установки изображений, отображаемых в действиях, нужно настроить свойство Images. Для этого щелкните два раза по компоненту (рисунок 5.9).

Для добавления действия нажмите клавишу Ins. В списке Actions появится новое действие. Для настройки свойств и событий действия используйте Область инспектора. Свойства, которые

можно настраивать у действия, представлены в таблице 5.20. События действий описаны в таблице 5.21.

Таблица 5.20 - Свойства действий

Свойство	Описание
Caption	Отображаемое название действия
Name	Имя действия
ImageIndex	Номер рисунка, ассоциируемого с действием
Visible	Видимость действия

Таблица 5.21 - События действий

Событие	Описание
OnExecute	Событие, происходящее при выполнении действия
OnHint	Событие, происходящее при отображении всплывающей подсказки

Визуальные компоненты, используемые в примере

TBitBtn представляет собой кнопку с изображением. Свойства компонента представлены в таблице 5.22, событие компонента – в таблице 5.23.

TDBGrid – компонент, отображающий сетку строк. Предназначен для отображения набора данных. События TDBGrid представлены в таблице 5.24.

Таблица 5.22 - Свойства компонента TBitBtn

Свойства	Описание
Caption	Надпись, отображаемая на кнопке
ModalResult	Значение, которое записывается в результат отображения модальной формы
Glyph	Изображение, отображаемое на кнопке

Таблица 5.23 - Событие компонента TBitBtn

Событие	Описание
OnClick	Событие, происходящее по нажатию на кнопку

Таблица 5.24 - События компонента TDBGrid

Событие	Описание
Align	Выравнивание компонента по форме
Columns	Настройка визуальных свойств (название, шрифт и т. д.) полей, отображаемых в сетке строк
DataSource	Источник данных для отображения данных в сетке таблицы

Таблица 5.25 - Свойства компонента TLabel

Свойство	Описание
Caption	Отображаемая надпись
Font	Шрифт отображаемой надписи
WordWrap	Перенос по слогам

Таблица 5.26 - Свойства компонента TEdit

Свойство	Описание
Text	Отображаемый текст

Таблица 5.27 - События компонента TEdit

Событие	Описание
OnChange	Изменение надписи в компоненте
OnClick	Щелчок левой кнопкой мышки по компоненту

Таблица 5.28 - Свойства компонента TComboBox

Свойство	Описание
Text	Отображаемое значение
Items	Содержит значения выпадающего списка
ItemIndex	Номер отображаемого значения из выпадающего списка
Style	Стиль отображения значения выпадающего списка

Таблица 5.29 - События компонента TComboBox

Событие	Описание
OnChange	Изменение надписи в компоненте
OnClick	Щелчок левой кнопкой мышки по компоненту
OnSelect	Выбор значения из выпадающего списка

Таблица 5.30 - Свойства компонента TPageControl

Свойство	Описание
Align	Способ расположения компонента на форме
ActivePage	Свойство определяет текущую страницу
ActivePageIndex	Свойство определяет номер текущей страницы
Caption	Название листа

Таблица 5.31 - Событие компонента TPageControl

Событие	Описание
OnChange	Изменение текущей страницы

TLabel – компонент, отображающий нередактируемую строку. Свойства TLabel представлены в таблице 5.25.

TEdit – компонент, отображающий редактируемую надпись. Свойство TEdit представлено в таблице 5.26, события данного компонента приведены в таблице 5.27.

TComboBox – компонент, представляющий собой комбинированный выпадающий список. Свойства компонента представлены в таблице 5.28, его события – в таблице 5.29.

TPageControl – компонент, содержащий список страницы и закладок, с помощью которых можно переходить с одной страницы на другую. После добавления компонента на форму для добавления новой страницы нужно вызвать контекстное меню, в котором выбрать пункт New Page. Свойства компонента TPageControl представлены в таблице 5.30, событие компонента TPageControl приведены в табл. 5.31.

TPanel – компонент, предназначенный для организации расположения других визуальных компонентов на форме. Выравнивание компонента TPanel настраивается в свойстве Align.

Разработка интерфейса

Добавление новой формы осуществляется путем выбора в главном меню пункта File → New → Form – Delphi. При этом будет создана новая форма, которая появится на экране. При добавлении новой формы в проект она встает в список автоматически создаваемых форм.

Для удаления формы из этого списка нужно в главном меню выбрать пункт Project → Options. Появится диалог, представленный на рисунке 5.10. Для удаления формы из списка автоматически создаваемых форм нужно выбрать ее в списке Auto-create form и нажать кнопку >.

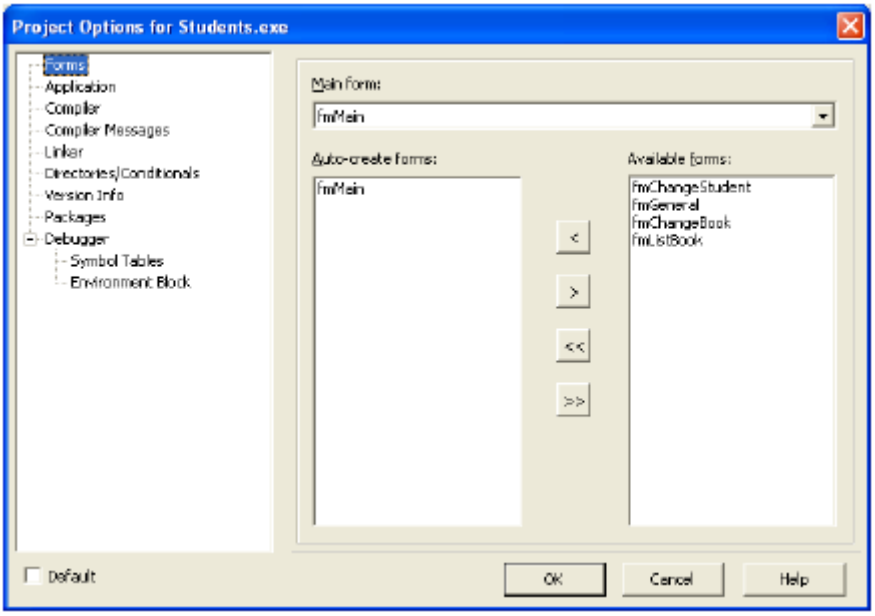


Рисунок 5.10 - Окно свойства проекта

Пример создания интерфейса

Создадим интерфейс, обладающий следующей функциональностью:

- редактирование списка студентов;
- редактирование списка книг;
- выдача книг студентам.

Для этого создадим четыре формы.

Главная форма

Главная форма является основной рабочей формой всей программы.

Пример интерфейса представлен на рисунок 5.11 и рисунок 5.12.

Для создания главной формы выполните следующие шаги:

1. Поместите на форму компонент TPageControl, настройте у него свойство Align равным alClent – компонент займет все свободное место на форме.

2. Щелкните правой клавишей мышки по компоненту TPageControl. Появится контекстное меню, в котором выберите пункт New Page. Будет создана новая страница.

3. Повторите действие п. 2 для создания еще одной страницы.

4. Добавьте на форму компонент TDBGrid, настройте у него свойство Align равным alTop – компонент займет верхнюю часть страницы. Так же необходимо настроить свойство Heigth – высота компонента.

5. Щелкните два раза по компоненту TDBGrid – появится новое окно (рисунок 5.13).

6. Для добавления поля нужно вызвать контекстное меню, в котором следует выбрать пункт Add или нажать кнопку Ins.

7. Выберите созданное поле и в Области инспектора настройте свойства Caption (название столбца) и Fieldname (название отображаемого поля).

8. Повторите п. 6–7 столько раз, сколько необходимо создать полей.

9. Добавьте на форму компонент TPanel, настройте свойство Align равным alTop – компонент займет место сразу за компонентом TDBGrid. Очистите свойство Caption у компонента TPanel.

10. Добавьте на форму компонент TPanel, настройте свойство Align равным alButom – компонент займет место внизу страницы. Очистите свойство Caption у компонента TPanel.

11. Добавьте еще один компонент TDBGrid, настройте свойство Align равным alClient – компонент займет все свободное место на форме.

12. Настройте у компонента, добавленного в п. 11, столбцы так же, как описано в п. 5–8.

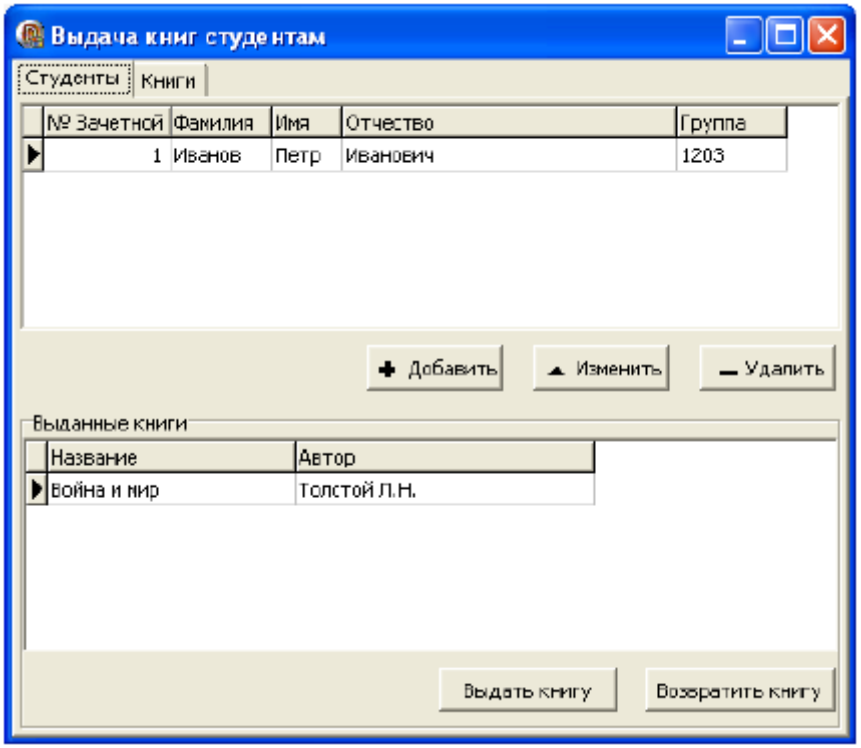


Рисунок 5.11 - Интерфейс выдачи студентам книг

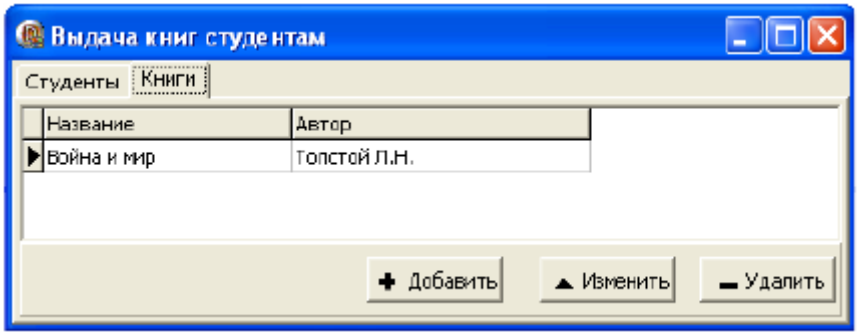


Рисунок 5.12 - Интерфейс редактирования списка книг

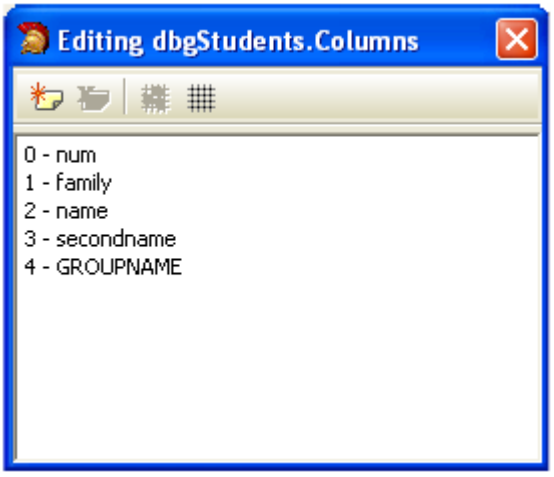


Рисунок 5.13 - Редактор отображаемых полей

13. Добавьте несколько компонентов TBitBtn на компоненты TPanel согласно рисунку 5.11 и 5.12.

14. Для добавления пиктограмм в проект нужно добавить компонент TImageList. Добавление пиктограммы осуществляется путем двойного щелчка по компоненту TImageList – появится окно настройки изображений (рисунок 5.8). В этом окне нажмите кнопку Add, в появившемся диалоге выберите графический файл пиктограммы.

15. Для реализации действий нужно добавить компонент TActionManager. Настройте свойство Images, хранящее пиктограммы.

16. Для добавления действий нужно два раза щелкнуть по компоненту TActionManager – появится редактор действий (рисунок 5.9).

17. Для добавления действия нужно нажать клавишу Ins.

18. Для настройки свойства действия выберите его в редакторе действий – его свойства отобразятся в Области инспектора. Свойства, которые нужно настроить, представлены в таблице 5.32.

Таблица 5.32 - Свойства действия

Свойство	Описание
Caption	Отображаемое название
Category	Категория действия
Name	Имя действия
ImageIndex	Номер пиктограммы из компонента TImageList

19. Также необходимо настроить событие, происходящее при выполнении действия. Для этого в Области инспектора выберите вкладку Events и два раза щелкните по событию OnExecute – появится редактор кода, в котором нужно записать требуемый код.

20. Нужно создать столько действий, сколько кнопок на форме.

21. Для назначения действия кнопки нужно выделить кнопку и в Области инспектора настроить свойство Action, выбрав то действие, которое должно происходить при нажатии кнопки.

Настройка интерфейса остальных форм выполняется по этому же принципу.

Форма редактирования параметров студента

Эта форма должна позволять пользователю задавать параметры студентов при добавлении новой записи и редактировании старой (см. рисунок 5.14).

Форма редактирования параметров книг

Форма должна позволять пользователю задавать параметры книги при редактировании и изменять параметры существующих книг (см. рисунок 5.15).

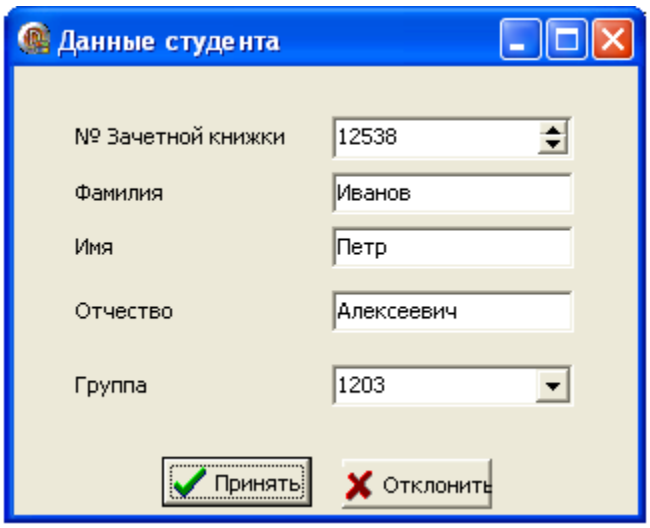


Рисунок 5.14 - Форма редактирования параметров студента

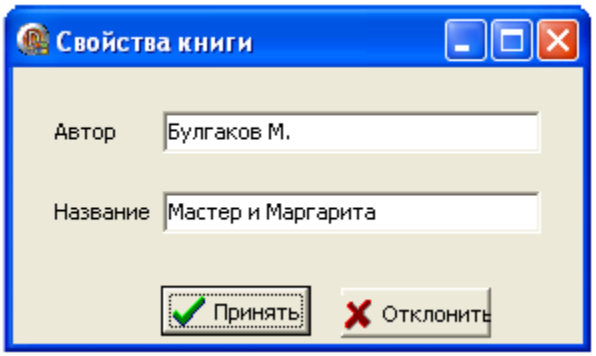


Рисунок 5.15 - Форма редактирования параметров книг

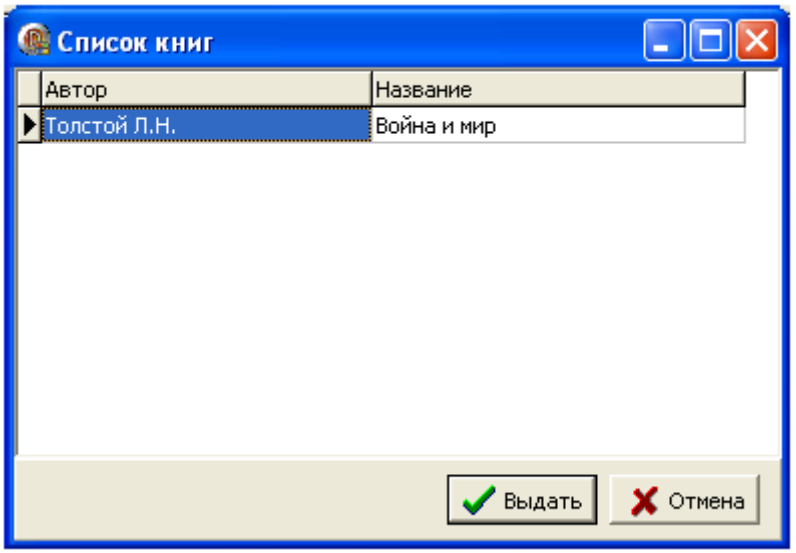


Рисунок 5.16 - Форма отображения списка книг

Форма отображения списка книг

Форма должна отображать список всех книг, для того чтобы можно было выбрать книгу при выдаче ее студенту (см. рисунок 5.16).

После создания форм необходимо подключить классы. Для этого нужно их создать для дальнейшего использования.

Подключение классов

После создания форм к интерфейсу нужно подключить классы. Для этого в главной форме проекта реализуем два события:

OnCreate. По этому событию создается объект TConnection, выполняется подключение к БД и создаются объекты класса бизнес-логики;

OnClose. По этому событию удаляются все созданные объекты.

Для создания событий:

в Конструкторе форм выберите объект, для которого вы хотите создать событие;

откройте вкладку Events в Области инспектора;

найдите требуемое событие и дважды щелкните по нему. При этом откроется редактор исходного кода на месте только что созданного события.

Пример кода для события On Create

```
var
//Объявление переменной класса по работе с ini-файлами
ini: tinifile;
// Объявление переменных для хранения параметров подключения
servername, path: string;
begin
 try
   //Подключение к ini–файлу
ini:=TIniFile.Create(ExtractFilePath(Application.ExeName)+'
```

```
config.ini');
// Чтение имени сервера, к которому необходимо подключиться
   if ini.ReadString('options','servername','')='' then
   begin
     servername:='localhost';
//Отображение диалога ввода имени сервера
     if  InputQuery('Введите  имя  сервера','',servername)
then
     begin
// Запись имени сервера в ini-файл
       ini.WriteString('options','servername',servername);
     end;
   end;
// Чтение имени сервера
   servername:=ini.ReadString('options','servername','');
// Чтение пути к файлу БД, если он пуст
   if ini.ReadString('options','path','')='' then
   begin
// Вывод диалога ввода пути к файлу БД
     if InputQuery('Введите путь к файлу БД','',path) then
     begin
// Запись пути к файлу БД в ini-файл
       ini.WriteString('options','path',path);
     end;
   end;
// Чтение пути к файлу БД из ini-файла
   path:=ini.ReadString('options','path','');
   try
// Создание объекта подключения
     fConnection:=TConnection.Create;
// Установка параметров подключения
     fConnection.ServerName:=servername;
     fConnection.DataBaseName:=path;
     fConnection.UserName:='SYSDBA';
     fConnection.Password:='masterkey';
//                  Подключение                к                  БД
  fConnection.Connect;
   except
// В случае установки неправильных параметров подключения возникнет
исключительная ситуация
     ShowMessage('Неправильные параметры подключения');
// Освобождение объекта подключения
     freeandnil(fConnection);
// Закрытие формы приложения
     fmMain.Close;
// Закрытие приложения
     Application.Terminate;
// Выход из события
     Exit;
   end;
 finally
// Удаление из памяти переменной класса по работе с ini-файлом
   FreeAndNil(ini);
 end;
// Создание класса слоя бизнес–логики по работе со студентами
 fStudents:=TStudents.Create(fConnection,fConnection.TransactionObject);
// Установка источника данных для сетки строк, отображающей список студентов
```

```
  dbgStudents.DataSource:=fStudents.DataSource;
// Выбор данных
  fStudents.Select;
// Создание класса слоя бизнес-логики по работе с книгами
fBooks:=TBook.Create(fConnection,fConnection.TransactionObject);
// Установка источника данных для сетки строк, отображающей список книг
  dbgBooks.DataSource:=fBooks.DataSource;
// Выбор данных
  fBooks.Select;
// Создание класса слоя бизнес-логики по работе с выданными
книгами
     fStudentBook :=TStudentBook.Create(fConnection, fConnection.TransactionObject);
// Установка события, происходящего при перемещении между записями
  fStudents.OnAfterScroll:=StudentsAfterScroll;
// Загрузка параметров выбранного студента в поля объекта
  fStudents.LoadCurrent;
// Установка полей объекта
  fStudentBook.fk_student:=fStudents.id;
// Выбор данных
  fStudentBook.Select;
// Установка источника данных для сетки строк, отображающей список выданных книг
  dbgStudentsBook.DataSource:=fStudentBook.DataSource;
end;
```

Для каждого действия в компоненте TActionManager необходимо реализовать событие OnExecute, в котором будет происходить вызов классов слоя бизнес-логики.

Пример реализации события On Execute

```
Try
// Загрузка параметров выбранного студента в поля объекта
   fStudents.LoadCurrent;
// Создание формы
  fmChangeStudent:=TfmChangeStudent.Create(Application);
    with fmChangeStudent do
    begin
// Заполнение значений компонентов на форме
      seNum.Value:=fStudents.Num;
      edFamily.Text:=fStudents.Family;
      edName.Text:=fStudents.Name;
      edSecondName.Text:=fStudents.SecondName;
// Заполнение списка, отображаемого в компоненте TComboBox
      cbGroup.Items.AddStrings(fStudents.GroupList);
// Установка значения, отображаемого в компоненте TComboBox
cbGroup.ItemIndex:=cbGroup.Items.IndexOf(fStudents.Group);
    end;
// Если пользователь нажал кнопку «Принять»
    if fmChangeStudent.ShowModal=mrOk then
    begin
// Заполнение полей объекта
      with fStudents do
      begin
        Num:=fmChangeStudent.seNum.Value;
                                          Family:=fmChangeStudent.edFamily.Text;
        Name:=fmChangeStudent.edName.Text;
        SecondName:=fmChangeStudent.edSecondName.Text;
        Group:=fmChangeStudent.cbGroup.Text;
// Вызов метода, обновляющего запись в таблице
        Update;
```

```
    end;
  end;
 finally
//Удаление из памяти формы
  FreeAndNil(fmChangeStudent);
 end;
```

Сохранение проекта

Для того чтобы сохранить проект, нужно в главном меню выбрать File → Save All. При нажатии этой кнопки появится диалог сохранения файла, в котором нужно указать имя и место расположения файла. Рекомендуется файл проекта называть исходя из названия предметной области (Students, Users, Library), файлы форм называть по функциональности формы, используя при этом префикс fm (fmUser, fmBook, fmGroup), файлы классов называть по именам классов, используя при этом префикс Class (ClassUser, ClassBook, ClassGroup).

Задание

Разработать приложение в Borland Developer Studio в соответствии с выбранной тематикой. Приложение должно работать с реляционной СУБД, например FireBird или Access), а его архитектура – соответствовать трехслойной архитектуре на базе объектно-реляционного отображения с типизированными объектами.

Контрольные вопросы

1. Что такое трехслойная архитектура?
2. Структура проекта Borland Developer Studio.
3. Отношения между классами в Developer Studio.
4. Типовая структура слоя реляционного отображения.
5. Классы слоя бизнес-логики, их особенности.
6. Типизированные объекты.
7. Разработка классов с использованием редактора UML. Достоинства, недостатки.
8. Заполнение данными компонента TDBGrid.
9. Работа с данными через компонент TComboBox.

Работа с литературой:

Рекомендуемые источники информации (№ источника)			
Основная	Дополнительная	Методическая	Интернет-ресурсы
1-3	1-3	1-3	1-2

Оценочные средства: контрольные вопросы

Тема 4. Архитектуры программных систем.

Лабораторная работа № 6. Реализация архитектуры на базе объектно-реляционного отображения с нетипизированными объектами

Форма проведения: Практическое выполнение задания

Цели работы:

- научиться использовать шаблоны проектирования;
- разработать структуру классов трехслойного приложения, используя методологию проектирования, основанную на шаблонах;
- создать класс согласно шаблону для обработки системных событий нетипизированными объектами;
- создать приложение БД, использующее трехслойную архитектуру на базе объектно-реляционного отображения с нетипизированными объектами.

Краткие теоретические сведения

Трехслойная архитектура на базе объектно-реляционного отображения с нетипизированными объектами

Понятие «нетипизированные объекты» связано со слоем графического интерфейса. Классы этого слоя могут работать с классами слоя бизнес-логики только посредством контроллера. Таким образом, интерфейс вызывает методы класса контроллера, а класс контроллера вызывает методы классов слоя бизнес-логики. Интерфейс не связан с классами

бизнес-логики, что позволяет менять интерфейс, не переписывая вызовы методов слоя бизнеслогики (рисунок 6.1).

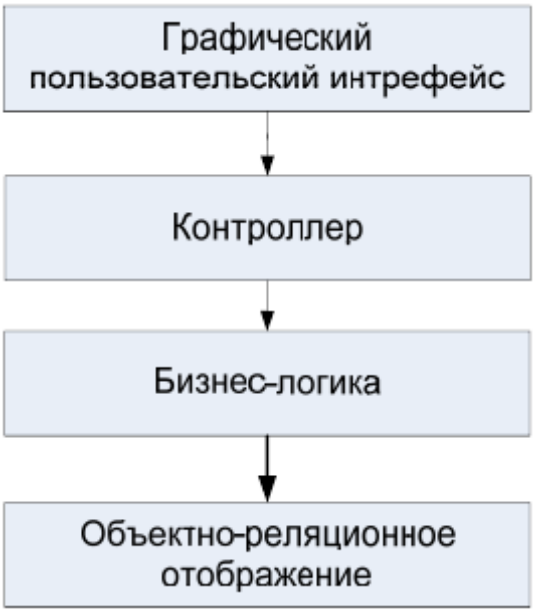


Рисунок 6.1 - Трехслойная архитектура с контроллером

Для разработки приложения на базе архитектуры с нетипизированными объектами мы будем применять шаблоны проектирования. Ниже описана часть основных шаблонов проектирования, используемых при разработке программных систем.

Шаблоны проектирования

Опытные разработчики объектно-ориентированных систем сформулировали общие принципы и стандартные решения, помогающие в разработке программного обеспечения. Если эти принципы и идиомы систематизировать и структурировать, а также присвоить им имена, то их можно применять в качестве шаблонов (patterns).

Шаблоны проектирования упрощают повторное использование удачных проектных и архитектурных решений. Представление прошедших проверку временем методик в виде паттернов проектирования облегчает доступ к ним со стороны разработчиков новых систем. С помощью шаблонов проектирования можно улучшить качество документации и сопровождения существующих систем, что позволит описать взаимодействия классов и объектов, а также причины, по которым система была построена так, а не иначе. Проще говоря, шаблоны проектирования дают разработчику возможность быстрее найти правильный путь.

В объектно-ориентированной технологии шаблоном проектирования называют именованное описание проблемы и ее решения, которые можно применить при разработке других систем. В идеале, шаблон должен содержать советы по его применению в различных ситуациях, а также описание его преимуществ и недостатков. Многие шаблоны содержат рекомендации по распределению обязанностей между объектами с учетом специфики задачи. Таким образом, шаблон – это именованная пара «проблема/решение», содержащая рекомендации по применению в конкретных ситуациях, которую можно использовать в различных контекстах.

Шаблон можно назвать новым, если он описывает новую идею. Однако сам термин «шаблон» означает стандартную, повторяющуюся сущность. Шаблоны не предназначены для изучения и выражения новых принципов разработки программного обеспечения, скорее, наоборот. Они призваны систематизировать существующие знания, идиомы и принципы: чем шире они используются, тем лучше.

Паттерн состоит из четырех основных элементов:

- имя. Всем шаблонам должны быть присвоены осмысленные имена. Именование шаблонов, методов и принципов имеет следующие преимущества: позволяет зафиксировать понятие в памяти человека, облегчает общение;

- задача. Этот элемент описывает, где следует применять паттерн. Необходимо сформулировать задачу и ее контекст. Может описываться конкретная проблема проектирования, например способ представления алгоритмов в виде объектов;
- решение. Описание элементов дизайна, отношений между ними, функций каждого элемента;
- результаты. Это следствия применения паттерна и разного рода компромиссы.

Шаблон Information Expert (информационный эксперт)

Проблема. Каков наиболее общий принцип распределения обязанностей между объектами при объектно-ориентированном проектировании?

Решение. Назначить обязанность информационному эксперту – классу, у которого имеется информация, требуемая для выполнения обязанности.

В модели системы могут быть определены десятки или сотни программных классов, а в приложении может потребоваться выполнение сотен или тысяч обязанностей. Во время объектно-ориентированного проектирования при формулировке принципов взаимодействия объектов необходимо распределить обязанности между классами. При правильном выполнении этой задачи система становится гораздо проще для понимания, поддержки и расширения. Кроме того, появляется возможность повторного использования уже разработанных компонентов в последующих приложениях.

При распределении обязанностей шаблон Information Expert используется гораздо чаще любого другого шаблона. В нем определены основные принципы, которые уже давно используются в объектно-ориентированном проектировании. Шаблон Expert не содержит неясных или запутанных идей и отражает обычный интуитивно понятный подход. Он заключается в том, что объекты осуществляют действия, связанные с имеющейся у них информацией.

Обратите внимание, что для выполнения обязанности зачастую требуется информация, распределенная между различными классами или объектами. Это предполагает, что должно существовать множество «частичных» экспертов, взаимодействующих при выполнении общей задачи. Например, для вычисления общей суммы продажи, в конечном счете, необходимо взаимодействие трех классов. Если информация распределена между различными объектами, то при выполнении общей задачи они должны взаимодействовать с помощью сообщений.

Преимущества

1. Шаблон Expert поддерживает инкапсуляцию. Для выполнения требуемых задач объекты используют собственные данные.
2. Соответствующее поведение системы обеспечивается несколькими классами, содержащими требуемую информацию. Это приводит к определениям классов, которые гораздо проще понимать и поддерживать.

Шаблон Creator (создатель)

Проблема. Кто должен отвечать за создание нового экземпляра некоторого класса?

Решение. Назначить классу B обязанность создавать экземпляры класса A, если выполняется одно из следующих условий:

- Класс B агрегирует объекты A;
- Класс B содержит объекты A;
- Класс B активно использует объекты A;
- Класс B обладает данными инициализации, которые будут передаваться объектам A при их создании (т. е. при создании объектов A класс B является экспертом);
- Класс B – создатель объектов A.

Если выполняется несколько из этих условий, то лучше использовать класс B, агрегирующий или содержащий класс A.

Создание объектов в объектно-ориентированной системе является одним из наиболее стандартных видов деятельности. Следовательно, при назначении обязанностей, связанных с созданием объектов, полезно руководствоваться некоторым основным принципом. Правильно распределив обязанности при проектировании, можно создать слабо связанные независимые компоненты с возможностью их дальнейшего использования, упростить их, а также обеспечить инкапсуляцию данных и их повторное использование.

Рассмотрение и распределение обязанностей выполняют в процессе создания диаграммы взаимодействий. Затем полученные результаты могут быть реализованы в конкретных методах, помещенных в разделе методов диаграммы классов.

Шаблон Creator определяет способ распределения обязанностей, связанный с процессом создания объектов. В объектно-ориентированных системах эта задача является одной из наиболее распространенных. Основным назначением шаблона Creator является выявление объекта-создателя, который при возникновении любого события должен быть связан со всеми созданными им объектами. При таком подходе обеспечивается низкая степень связанности.

В некоторых случаях в качестве создателя выбирается класс, который содержит данные инициализации, передаваемые объекту во время его создания. На самом деле это пример использования шаблона Expert. В процессе создания инициализирующие данные передаются с помощью метода инициализации некоторого вида, такого как конструктор языка Java с параметрами.

Преимущества

Применение шаблона Creator не повышает степени связанности, поскольку созданный (created) класс, как правило, оказывается видимым для класса-создателя посредством имеющихся ассоциаций.

Шаблон Low Coupling (слабое связывание)

Проблема. Как обеспечить зависимость, незначительное влияние изменений и повысить возможность повторного использования?

Решение. Распределить обязанности таким образом, чтобы степень связанности оставалась низкой.

Степень связанности (coupling) – это мера, определяющая, насколько жестко один элемент связан с другими элементами либо каким количеством данных о других элементах он обладает. Элемент с низкой степенью связанности (или слабым связыванием) зависит от не очень большого числа других элементов. Выражение «очень много» зависит от контекста, однако необходимо провести его оценку.

Класс с высокой степенью связанности (или жестко связанный) зависит от множества других классов. Однако наличие таких классов нежелательно, поскольку оно приводит к возникновению следующих проблем:

- изменения в связанных классах приводят к локальным изменениям в данном классе;
- затрудняется понимание каждого класса в отдельности;
- усложняется повторное использование, поскольку для этого требуется дополнительный анализ классов, с которыми связан данный класс.

В шаблоне Low Coupling описывается принцип, о котором нельзя забывать на протяжении всех стадий работы над проектом. Он является объектом постоянного внимания. Шаблон Low Coupling представляет собой средство, которое разработчик применяет при оценке всех принимаемых в процессе проектирования решений.

Шаблон Low Coupling поддерживает независимость классов, что, в свою очередь, повышает возможности повторного использования и обеспечивает более высокую эффективность приложения. Его нельзя рассматривать изолированно от других шаблонов, таких как Expert и High Cohesion. Скорее, он обеспечивает один из основных принципов проектирования, применяемых при распределении обязанностей.

Подкласс жестко связан со своим суперклассом. Поэтому, принимая решение о наследовании свойств объектов, следует учитывать, что отношение наследования повышает степень связанности классов.

Не существует абсолютной меры для определения слишком высокой степени связывания. Важно лишь понимать степень связанности объектов на текущий момент и не упустить тот момент, когда дальнейшее повышение степени связанности может привести к возникновению проблем. В целом, следует руководствоваться таким принципом: классы, которые являются достаточно общими по своей природе и с высокой вероятностью будут повторно использоваться в дальнейшем, должны иметь минимальную степень связанности с другими классами.

Крайним случаем при реализации шаблона Low Coupling является полное отсутствие связывания между классами. Такая ситуация тоже нежелательна, поскольку базовой идеей объектного подхода является система связанных объектов, которые «общаются» между собой посредством передачи сообщений. При слишком частом использовании принципа слабого связывания система будет состоять из нескольких изолированных сложных активных объектов, самостоятельно выполняющих все операции, и множества пассивных объектов, основная функция которых сводится к хранению данных. Поэтому при создании объектно-ориентированной системы

должна присутствовать некоторая оптимальная степень связывания между объектами, позволяющая выполнять основные функции посредством взаимодействия этих объектов.

Преимущества

1. Изменения компонентов мало сказываются на других объектах.
2. Принципы работы и функции компонентов можно понять, не изучая другие объекты.
3. Удобство повторного использования.

Шаблон High Cohesion (высокое зацепление)

Проблема. Как обеспечить возможность управления сложностью?

Решение. Распределение обязанностей, поддерживающее высокую степень зацепления.

Зацепление (функциональное зацепление) – это мера связанности и сфокусированности обязанностей класса. Считается, что элемент обладает высокой степенью зацепления, если его обязанности тесно связаны между собой и он не выполняет непомерных объемов работы. В роли таких элементов могут выступать классы, подсистемы и т. д.

Класс с низкой степенью зацепления выполняет много разнородных функций или несвязанных между собой обязанностей. Такие классы создавать нежелательно, поскольку они приводят к возникновению следующих проблем:

- трудность понимания назначения класса;
- сложности при повторном использовании;
- сложности поддержки;
- ненадежность, постоянная подверженность изменениям.

Классы со слабым зацеплением, как правило, являются слишком «абстрактными» или выполняют обязанности, которые можно легко распределить между другими объектами.

На практике уровень зацепления не рассматривают изолированно от других обязанностей и принципов, обеспечиваемых шаблонами Expert и Low Coupling.

Как и о принципе слабого связывания, о высокой степени зацепления следует помнить в течение всего процесса проектирования. Этот шаблон необходимо применять при оценке эффективности каждого проектного решения.

Следующие сценарии иллюстрируют различную степень функционального зацепления.

1. Очень слабое зацепление. Только один класс отвечает за выполнение множества операций в самых различных функциональных областях.
2. Слабое зацепление. Класс несет единоличную ответственность за выполнение сложной задачи из одной функциональной области.
3. Среднее зацепление. Класс имеет среднее количество обязанностей из одной функциональной области и для выполнения своих задач взаимодействует с другими классами.
4. Сильное зацепление. Класс имеет несложные обязанности в нескольких различных областях, логически связанных с концепцией этого класса, но не связанных между собой. Пример. Существует класс Company, который несет полную ответственность: за знание сведений обо всех сотрудниках компании; всю финансовую информацию. Эти две области не слишком связаны между собой, однако обе логически связаны с понятием «компания». К тому же предполагается, что такой класс содержит небольшое число открытых методов и требует незначительных объемов кода для их реализации.

Как правило, класс с высокой степенью зацепления содержит сравнительно небольшое число методов, которые функционально тесно связаны между собой, и не выполняет слишком много функций. Он взаимодействует с другими объектами для выполнения более сложных задач.

Преимущества

1. Повышаются ясность и простота проектных решений.
2. Упрощаются поддержка и доработка.
3. Зачастую обеспечивается слабое связывание.
4. Улучшаются возможности повторного использования, поскольку класс с высокой степенью зацепления выполняет конкретную задачу.

Шаблон Controller (контроллер)

Проблема. Кто должен отвечать за обработку входных системных событий?

Системное событие (system event) – это событие высокого уровня, генерируемое внешним исполнителем (событие с внешним входом). Системные события связаны с системными операциями (system operation), т. е. операциями, выполняемыми системой в ответ на события.

Решение. Делегирование обязанностей по обработке системных сообщений классу, удовлетворяющему одному из следующих условий:

- класс представляет всю систему в целом, устройство или подсистему (внешний контроллер);
- класс представляет сценарий некоторого прецедента, в рамках которого выполняется обработка всех системных событий, и обычно называется Handler, Coordinator или Session (контроллер прецедента или контроллер сеанса);
- для всех системных событий в рамках одного сценария прецедента используется один и тот же класс-контроллер.

Заметим, что в этот перечень не включаются классы, реализующие окно, аплет, приложение, вид и документ. Такие классы не выполняют задачи, связанные с системными событиями. Они обычно получают сообщения и делегируют их контроллерам.

Контроллер(controller) – это объект, не относящийся к интерфейсу пользователя и отвечающий за обработку системных событий. Контроллер определяет методы для выполнения системных операций.

Большинство систем получает внешние события. Обычно они связаны с графическим интерфейсом пользователя. Кроме того, системе могу передаваться внешние сообщения, например при обработке телекоммуникационных сигналов или сигналов от датчиков в системах управления.

Во всех случаях при использовании объектно-ориентированного подхода для обработки внешних событий применяются контроллеры. Шаблон Controller обеспечивает наиболее типичные проектные решения для этого случая. Как видно из рисунка 6.1, контроллер – это своеобразный вид интерфейса между уровнями предметной области и графического представления.

Типичной ошибкой при создании контроллеров является возложение на них слишком большого числа обязанностей. Обычно контроллер должен лишь делегировать функции другим объектам и координировать их деятельность, а не выполнять эти действия самостоятельно.

Контроллеры делятся на два типа.

1. Внешний контроллер(facade controller). Представляет всю систему, устройство или подсистему. Основная идея сводится к выбору некоторого класса, имя которого охватывает все слои приложения. Этот класс обеспечивает главную точку вызова всех служб из интерфейса пользователя и обращения к остальным слоям. Этот класс может представлять физический объект, телекоммуникационный переключатель, всю программную часть системы или любые другие понятия, выбранные разработчиком для представления системы в целом. Внешние контроллеры удобно использовать в том случае, когда существует лишь несколько системных событий или системные сообщения невозможно перенаправить другим контроллерам, наподобие системы обработки сообщений.
2. Контроллер прецедента (use case controller). Для каждого прецедента должен существовать отдельный контроллер. Контроллеры прецедентов следует использовать в том случае, когда применение внешнего контроллера приводит к слабой степени зацепления или высокой степени связывания. Контроллер прецедента вводится в том случае, если на существующий контроллер возлагается слишком много дополнительных обязанностей. Контроллеры прецедентов следует применять при наличии большого числа системных событий, распределенных между различными процессами.

Преимущества

1. Улучшение условий для повторного использования компонентов. Применение этого шаблона обеспечивает обработку процессов предметной области на уровне реализации объектов, а не на уровне интерфейса. Обязанности контроллера могут быть технически реализованы в объектах интерфейса, однако в этом случае программный код и логические решения, относящиеся к процессам предметной области, будут жестко связаны с элементами интерфейса, например с окнами. При этом снижается эффективность повторного использования компонентов в других приложениях, поскольку процессы предметной области ограничены рамками интерфейса (например, связаны с оконным объектом), что может оказаться неприемлемым в других приложениях. Делегирование выполнения системных операций специальному контроллеру облегчает повторное использование логики обработки подобных процессов в последующих приложениях.
2. Контроль состояния прецедента. Иногда необходимо удостовериться, что системные операции выполняются в некоторой определенной последовательности. Например, нужно гарантировать, чтобы операция makePayment выполнялась только после операции endSale, для

чего необходимо накапливать информацию о последовательности событий. Для этой цели удобно использовать контроллер, особенно контроллер прецедента.

Применение шаблона Information Expert

Шаблон информационный эксперт отвечает на вопрос, какой класс должен отвечать за ту или иную функциональность. Ответом на этот вопрос является то, что класс должен отвечать за ту или иную функциональность, в случае если у него есть достаточно данных для решения задач функциональности.

В примере из лабораторной работы №5 у нас есть два объекта взаимодействующих между собой – это Студенты и Книги. Очевидно, что у них есть определенная функциональность, эти объекты должны уметь хранить данные о себе, создаваться, изменяться и удаляться. Согласно шаблону Information Expert мы создаем эти классы (TSudents, TBook) и определяем эту функциональность. Теперь возьмем процесс выдачи книг и возврата их от студентов. Нам необходимо отображать список книг, выданных студенту, если мы добавим эту функциональность классу TStudents или классу TBook, то тогда мы их нагрузим лишними функциями. Оптимальным решением данной проблемы является созданием нового класса TStudentBook, в функциональность которого входят эти функции.

Применение шаблона Creator

Шаблон Creator отвечает на вопрос, кто должен создавать экземпляр нового объекта. Класс должен создавать экземпляр нового объекта, если выполняется одно или несколько условий:

- Класс B агрегирует объекты A;
- Класс B содержит объекты A;
- Класс B активно использует объекты A;
- Класс B обладает данными инициализации, которые будут передаваться объектам A при их создании (т. е. при создании объектов A класс B является экспертом);
- Класс B – создатель объектов A.

В качестве примера возьмем базовый класс для слоя бизнес-логики – класс TDataPrepare. Наследники этого класса должны обращаться к классу слоя объектно-реляционного отображения TExecuteObject для выполнения SQL-запросов. Существуют два вида запросов: возвращающие данные и не возвращающие данные. Запросы, возвращающие данные, используются для отображения одного или несколько объектов из реляционной таблицы. Запросы, не возвращающие данные, используются для модификации данных в реляционных таблицах. Таким образом, у нас есть проблема: кто должен создавать объект класса TExecuteObject. Поскольку разные объекты отображают данные из различных таблиц БД, то получается, что каждому объекту класса TDataPrepare необходимо иметь по одному экземпляру объекта TExecuteObject. Но тут появляется другая проблема при выполнении запросов, не возвращающих данные, текущий набор данных закрывается и при попытке другого запроса текущий набор данных закроется, и его необходимо будет заново открыть, что увеличит объем данных, передаваемых по сети. Тогда ответим на другой вопрос: кто должен отвечать за создание и удаления объекта, выполняющего такие запросы? Существуют два варианта: класс TDataPrepare или другой – новый класс. Решение данной проблемы зависит от того, могут ли быть отправлены одновременно два и более таких запросов. Если да, то тогда необходимо, чтобы этот объект создавал класс TDataPrepare, если нет, то тогда необходимо создать новый класс, обладающий такой функциональностью.

Использование шаблона High Cohesion

Шаблон High Cohesion позволяет управлять сложностью классов. Чем проще класс, тем проще его понять, использовать, модифицировать, документировать. Это не означает, что не должно быть сложных классов. Сложность должна быть достаточной для решения проблемы, не более.

В качестве примера возьмем процесс выдачи книг и возврата их от студентов. Нам необходимо:

- отображать список книг, выданных студенту;
- выдавать студенту новые книги;
- принимать от студента книги.

Если мы добавим эту функциональность классу TStudents или классу TBook, то тогда мы их нагрузим лишними функциями, поскольку эти классы могут существовать без этого процесса. Например, есть книги, которые не выдают студентам, и есть студенты, которые не берут книги в библиотеке.

Оптимальным решением данной проблемы является создание нового класса TStudentBook, в функциональность которого будет входить реализация этих функций. Данные, необходимые для процесса выдачи или возврата книг, нужно брать у классов TStudents и TBook. Таким образом, у нас появилось три класса, каждый из которых нагружен только ему присущей функциональностью.

Применение шаблонаController

Шаблон Controller позволяет централизованно обрабатывать системные события, возникающие в приложении. При использовании этого шаблона интерфейс приложения посылает сообщения классам слоя бизнес-логики через класс Contoller. Использование этого шаблона подразумевает создание одного или нескольких классов, которые обрабатывают системные события. Под системными событиями подразумевается функциональные действия пользователей.

Согласно шаблону Creator класс TController должен создавать и уничтожать объекты слоя бизнес-логики, поэтому необходимо в этом классе создать поля, представленные в таблице 6.1.

Таблица 6.1 - Поля класса TController

Название	Описание
fStudents: TStudents	Класс по работе со студентами
fBooks: TBook	Класс по работе с книгами
fStudentBook: TStudentBook	Класс, выдающий и забирающий книги

Согласно шаблону High Cohesion эти классы имеют высокое зацепление, поскольку каждый класс работает только с одним объектом в БД.

Для обработки входных системных событий необходимо создать методы, каждый из которых будет обрабатывать только одно системное событие (см. таблицу 6.2).

Таблица 6.2 - Методы класса TController

Название	Описание
AddStudent (num: integer; family, name, secondname, group: string)	Метод добавляет студента
AddBook (Author, Name: string)	Метод добавляет книгу
AddStudentBook	Метод выдает книгу студенту
UpdateStudent (num: integer; family, name, secondname, group: string)	Метод изменяет выбранного студента
UpdateBook (Author, Name: string)	Метод изменяет выбранную книгу
DeleteStudent	Метод удаляет выбранного студента
DeleteBook	Метод удаляет выбранную книгу
DeleteStudentBook	Метод принимает книгу от студента
StudentParams: tParams	Функция возвращает параметры текущего студента
BookParams: tParams	Функция возвращает параметры текущей книги
GroupList: TStringList	Функция возвращает список групп студентов
StudentsAfterScroll(DataSet: TDataSet)	Событие, происходящее при перемещении по набору данных, отображающих студентов

Для обеспечения отображения данных необходимо создать свойства, возвращающие ссылки на источники наборов данных (таблица 6.2). В результате диаграмма класса примет вид представленный, на рисунке 6.2.

Таблица 6.3 - Свойства TController

Название	Описание
StudentDataSource: TDataSource	Источник данных набора отображающих студентов
BookDataSource: TDataSource	Источник данных набора отображающих книги
StudentBookDataSource: TDataSource	Источник данных отображающий выданные книги

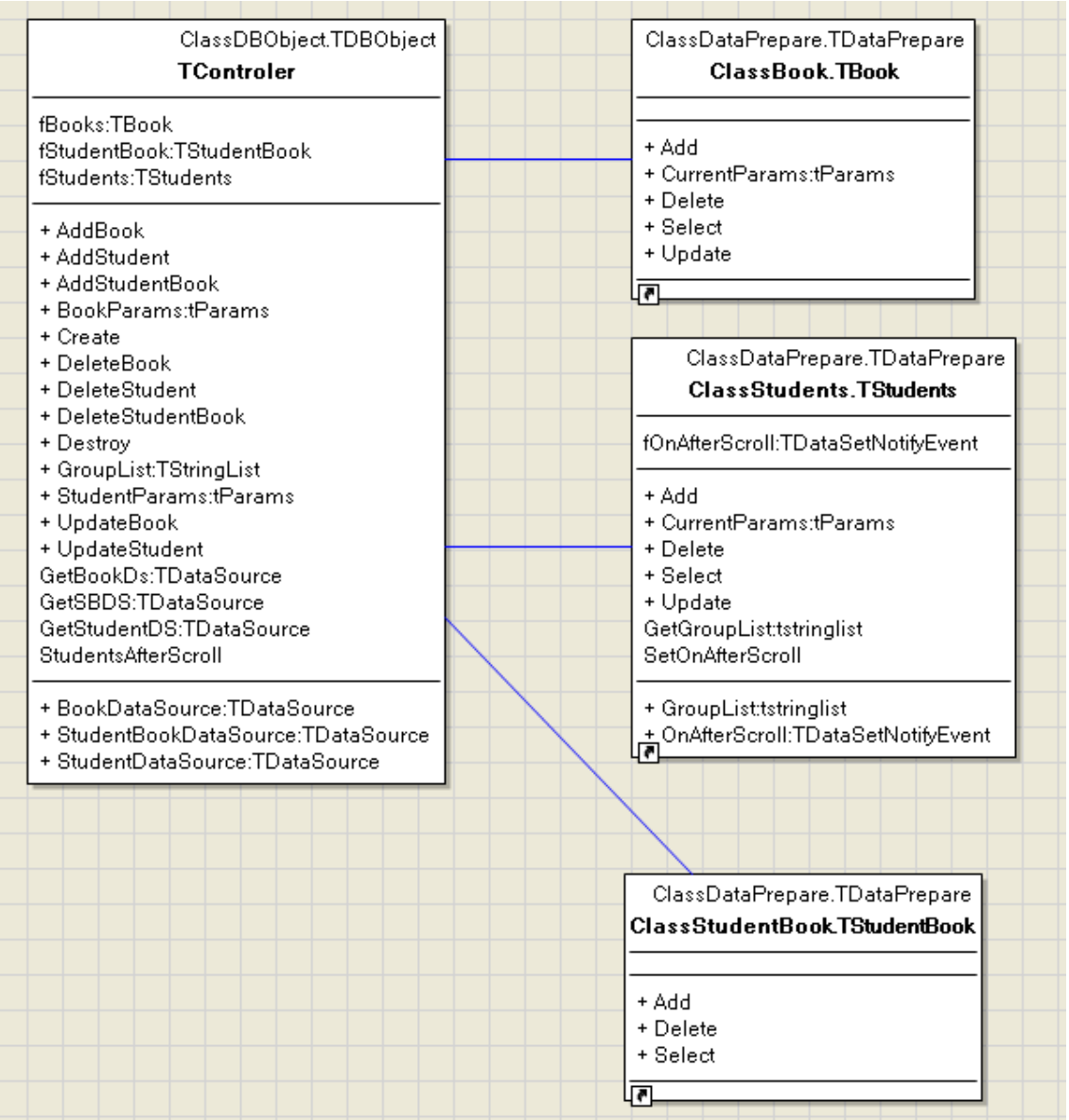


Рисунок 6.2 - Диаграмма классов слоя бизнес-логики

Задание и порядок выполнения работы

Разработать приложение в соответствии с тематикой. Приложение должно работать с реляционной СУБД, а его архитектура соответствовать трехслойной архитектуре на базе объектно-реляционного отображения с нетипизированными объектами. При разработке приложения необходимо применить шаблоны проектирования.

Контрольные вопросы

1. Что такое шаблон проектирования?
2. Какова структура шаблона проектирования?
3. Каков принцип разработки классов на основе шаблонов проектирования?
4. Шаблон проектирования Information Expert.
5. Шаблон проектирования Creator.
6. Шаблон проектирования Low Coupling.
7. Шаблон проектирования High Cohesion.
8. Шаблон проектирования Controller.
9. Чем отличается архитектура с типизированными объектами от архитектуры с нетипизированными объектами?

Работа с литературой:

Рекомендуемые источники информации (№ источника)

Основная	Дополнительная	Методическая	Интернет-ресурсы
1-3	1-3	1-3	1-2

Оценочные средства: контрольные вопросы

Тема 5. Технология MDA.

Лабораторная работа № 7. Разработка простого MDA-приложения

Форма проведения: Практическое выполнение задания

Цели работы:

- ознакомиться с архитектурой, управляемой моделью MDA, технологией ECO и языком OCL;
- научиться создавать простое приложение по технологии MDA.

Краткие теоретические сведения

Архитектура, управляемая моделью MDA

Концепция Model Driven Architecture (MDA) – *архитектура, управляемая моделью* – была разработана в независимой некоммерческой организации Object Management Group (OMG) – консорциум объектного управления.

Она объединяет сотни компаний-производителей программного и аппаратного обеспечения.

В технологии MDA основным элементом проектирования считается модель. *Модель – это описание реальной системы, учитывающее определенные характеристики или аспекты моделируемого объекта, процесса или явления.*

В конкретном проекте модель MDA представляет собой законченное и формализованное представление создаваемого программного продукта или его смысловой части. Высокая точность и непротиворечивость описания необходима, чтобы автоматизировать процесс перевода модели в программный (обычно исходный) код.

Понятие *«управление моделью»* подразумевает прямое использование модели при любых действиях по проектированию, разработке и развертыванию системы. При внесении изменений в архитектуру приложения модель считается первичной. Пусть, например, требуется пополнить описание класса новым полем или методом. В классическом (не модельном) подходе к разработке модификация описания класса выполнялась бы в исходном коде, ручным кодированием. В технологии MDA происходит модификация визуальной диаграммы, на которой класс представлен в виде графического элемента.

На базе такой диаграммы исходный код с измененным описанием класса генерируется автоматически.

Под *архитектурой, управляемой моделью,* понимается полная и законченная архитектура приложения, целиком формируемая с помощью модели.

В крупных проектах с моделью обычно работает не программист, а системный аналитик. Он отвлекается от мелких деталей реализации и перестает мыслить в терминах отдельных операторов языка программирования.

Хороший проектировщик способен охватить и продумать структуру больших функциональных логических блоков и основные взаимосвязи между ними.

Далее подобная модель детализируется и транслируется в исходный код на выбранном языке.

Технология MDA ориентирована на создание приложений, которые независимы от платформы, операционной системы и языков программирования. Она позволяет строить масштабируемые приложения из компонентов, которые могут использоваться повторно и многократно. Сам процесс разработки выполняется, как явствует из названия, под управлением модели.

Модель приложения – это взаимосвязанный набор визуальных диаграмм, наглядно описывающих внутреннюю структуру системы и принципы ее функционирования. Модель приложения не привязана к конкретному языку или конкретной среде программирования.

Визуальные диаграммы чаще всего строятся с помощью унифицированного языка моделирования UML.

Методология MDA описывает не столько моделирование, сколько метамоделирование, предусматривающее большую гибкость в конкретных, прикладных подходах к моделированию.

Метамоделирование – способ описания моделей, определяющий механизмы построения конкретных моделей программных систем с помощью базового словаря и набора ограничений, налагаемых на создаваемые модели.

Технология ECO

Корпорация Borland разработала собственную версию (реализацию) концепции MDA, которая называется Borland MDA (BMDA). Сегодня вместо термина BMDA применяется новый термин ECO. Технология ECO (*Enterprise Core Objects* – ключевые корпоративные объекты), реализующая концепцию MDA, стала наиболее важным улучшением последних версий среды Delphi. Она представлена в Delphi 2006 в виде третьей версии ECO III. Каждый компонент ECO представляет собой своеобразную программную обертку положений концепции MDA. Он является промежуточным слоем между средствами визуального проектирования программных моделей и их конкретной реализацией на языках программирования Delphi и C#.

Технология ECO переводит процесс согласования требований на модельный уровень. Диаграммы UML обычно хорошо воспринимаются и заказчиком, и исполнителем. Поэтому, хотя использование визуального языка моделирования в MDA не обязательно, почти 100 % проектов MDA создаются с применением языка UML.

Построение готового приложения происходит в несколько этапов. Сначала строятся платформно-независимые модели PIM (*Platform Independent Model*), затем выполняется их перенос в платформно-специфичные модели PSM (*Platform Specific Model*). Доступные на сегодняшний день продукты автоматизируют эти шаги на 50–70 %. Перенос моделей PSM в код конкретного языка программирования автоматизирован почти на 100 %.

Еще один подход, настоятельно рекомендуемый группой OMG при использовании подхода MDA, заключается в выделении логики приложения в отдельную, по возможности платформно-независимую, структуру. Достигается это использованием языка объектных ограничений OCL, который сегодня считается частью языка UML. Команды OCL встраиваются непосредственно в модель UML, описывая и уточняя аспекты ее работы. Удобство OCL еще и в том, что его выражения напоминают естественный язык (предложения, записанные на английском). Это позволяет сохранять при построении моделей контакт с людьми (например, представителями заказчика), не являющимися специалистами в области программирования.

Физические объекты, структура которых описана с помощью языка UML, а особенности существования – с помощью языка OCL, функционируют в процессе работы приложения в специально выделенной области оперативной памяти, так называемом *объектном пространстве.* Технология ECO способна автоматически отображать модель не только в программный код, но и в код на языке SQL. Он генерируется для построения схемы базы данных, хранящей копию объектного пространства. Поддержка разных СУБД является в ECO одной из ключевых технологий. С ее помощью удается сохранять текущее состояние объектного пространства приложения в базе данных и затем, после перезапуска приложения, загружать его и продолжать работу с прерванного места.

Среда Delphi на основе подготовленной модели автоматически формирует схему базы данных с учетом версии и специфики конкретной СУБД.

Она автоматически создает в ней все таблицы, нужные для хранения объектного пространства и взаимосвязей между классами, даже если это требует введения дополнительных таблиц. По мере совершенствования модели следует периодически синхронизировать схему базы данных с внесенными в модель модификациями, для этого достаточно нажать одну кнопку. Такой процесс называется в технологии ECO *эволюционным развитием базы данных.*

Стыковка объектного пространства с пользовательским интерфейсом реализуется в ECO с помощью *дескрипторов.* Дескриптор ECO – это стыковочная точка (компонент промежуточного уровня), связывающая с помощью выражений OCL элементы модели UML (классы) с программным кодом и внутренней структурой обычного приложения Delphi. Дескрипторы задают способы формирования наборов объектов в объектном пространстве по критериям, заданным выражениями OCL. Они также обеспечивают доступ к ним элементов пользовательского интерфейса, а программному коду предлагают ссылки на объекты ECO.

Как правило, ручное кодирование на языке Delphi требуется, если классы модели, построенной с помощью технологии ECO, включают методы, которые необходимо запрограммировать явно.

Серьезную роль в создании приложений ECO играет язык OCL. Он предоставляет программные средства обращения к пространству ECO и содержит средства создания,

модификации и уничтожения объектов, навигации по этому пространству и выполнения всевозможных вычислительных задач.

Язык объектных ограничений OCL

Архитектура MDA ставит на первое место модель приложения, и поэтому она непосредственно связана с языком, на котором такие модели создаются (язык унифицированного моделирования UML). Один из самых серьезных и справедливо критикуемых недостатков языка моделирования UML – предоставление разработчику только визуальных средств моделирования.

Эти средства абстрактны, поэтому далеко не всегда способны точно и формально отразить тот или иной нюанс функционирования проектируемой системы. Именно необходимость формализации описания условий и ограничений, накладываемых на элементы диаграмм классов, вызвало появление OCL (Object Constraint Language). Конечно, такие условия могут быть сформулированы и на естественном языке, однако он не будет являться строгим и может допускать неоднозначные трактовки.

Язык OCL не является языком программирования, т. е. не позволяет создать программу из своих операторов или описать логику выполнения каких-либо действий. Он создавался как формальный текстовый язык, дополняющий графические возможности языка UML. Выражение OCL обычно привязано к определенному классу и задает множество экземпляров этого класса. Команды OCL выполняют также фильтрацию этого множества или, например, определяют число его элементов.

Язык OCL содержит развитые средства манипуляции над множествами объектов, поэтому он хорошо подходит для решения задач, где обычно применяется язык запросов к реляционным базам данных SQL. Язык OCL позволяет организовывать запросы с большей эффективностью и на более высоком, модельном уровне абстракции, нежели язык SQL.

Выражение OCL фактически задает условие, которому должны удовлетворять все экземпляры соответствующего объекта UML. Результатом выполнения выражения OCL является множество объектов, удовлетворяющих этому условию.

При этом средствами языка OCL невозможно изменить сами элементы модели (классы, атрибуты, отношения). Среда, вычисляющая выражение OCL, просто определяет результирующее значение, которое может впоследствии использоваться (хотя это и не обязательно).

Язык OCL был разработан в корпорации IBM, в 1997 году вышла спецификация языка версии 1.1, в разработке и согласовании которой приняли участие такие компании, как Rational, Microsoft, Oracle, Hewlett-Packard и ряд других. Сильной стороной языка OCL оказалась независимость от платформы реализации и легкая адаптация к разным средствам программирования.

MDI-контейнеры

MDI – Multiple Document Interface (многодокументный интерфейс). В приложениях с MDI в основном (родительском) окне можно открыть более одного дочернего окна. Данная возможность обычно используется в электронных таблицах или текстовых редакторах.

Каждое MDI приложение имеет три основные составляющие:

- одну (и только одну) родительскую форму MDI;
- одну и более дочерних форм MDI;
- основное меню MDI.

Родительская форма должна быть стартовой, она нуждается, по крайней мере, в одной дочерней форме. Дочерние формы MDI – это простые формы, за исключением того, что их видимая часть ограничена размерами родительского окна. При минимизации такого окна оно помещается не в панель задач, а остается внутри родительского окна (на панель задач попадет только родительское окно).

Создание простого MDA-приложения

Рассмотрим пример создания простого MDA-приложения. Имеются две сущности Кафедра и Преподаватель. Представим экземпляры этих сущностей в таблицах формы. В этих же таблицах можно будет модифицировать значения отдельных полей представленных в них объектов (экземпляров Кафедра и Преподаватель).

Для разработки приложения применятся среда разработки Borland Developer Studio 2006. Приложение разрабатывается для платформы .NET Framework.

Основные этапы разработки приложения

1. Формируется модель UML будущего приложения. Создаются классы, описываются их атрибуты и методы, настраиваются взаимосвязи между ними.

2. В проект добавляются и настраиваются невизуальные компоненты, связывающие созданную модель UML с прикладной частью проекта.

3. Пространство ECO связывается с базой данных. В ней будет долговременно храниться его копия: все объекты ECO и взаимосвязи между ними.

4. Проектируется пользовательский интерфейс. Задействуются компоненты, обеспечивающие связь интерфейса с моделью UML.

5. Создается переносимая логика приложения на языке OCL. Элементы управления связываются с выражениями OCL для выполнения типичных стандартных действий (добавление, редактирование и удаление объектов ECO).

Обзор возможностей Borland Developer Studio 2006 для разработки MDA-приложения

1. После запуска BDS 2006 на экране открывается главное окно среды разработки (рисунок 7.1). Проект Delphi представляет собой группу файлов, содержащих исходный текст программы и вспомогательные данные. Все они объединены тематикой решаемой задачи.

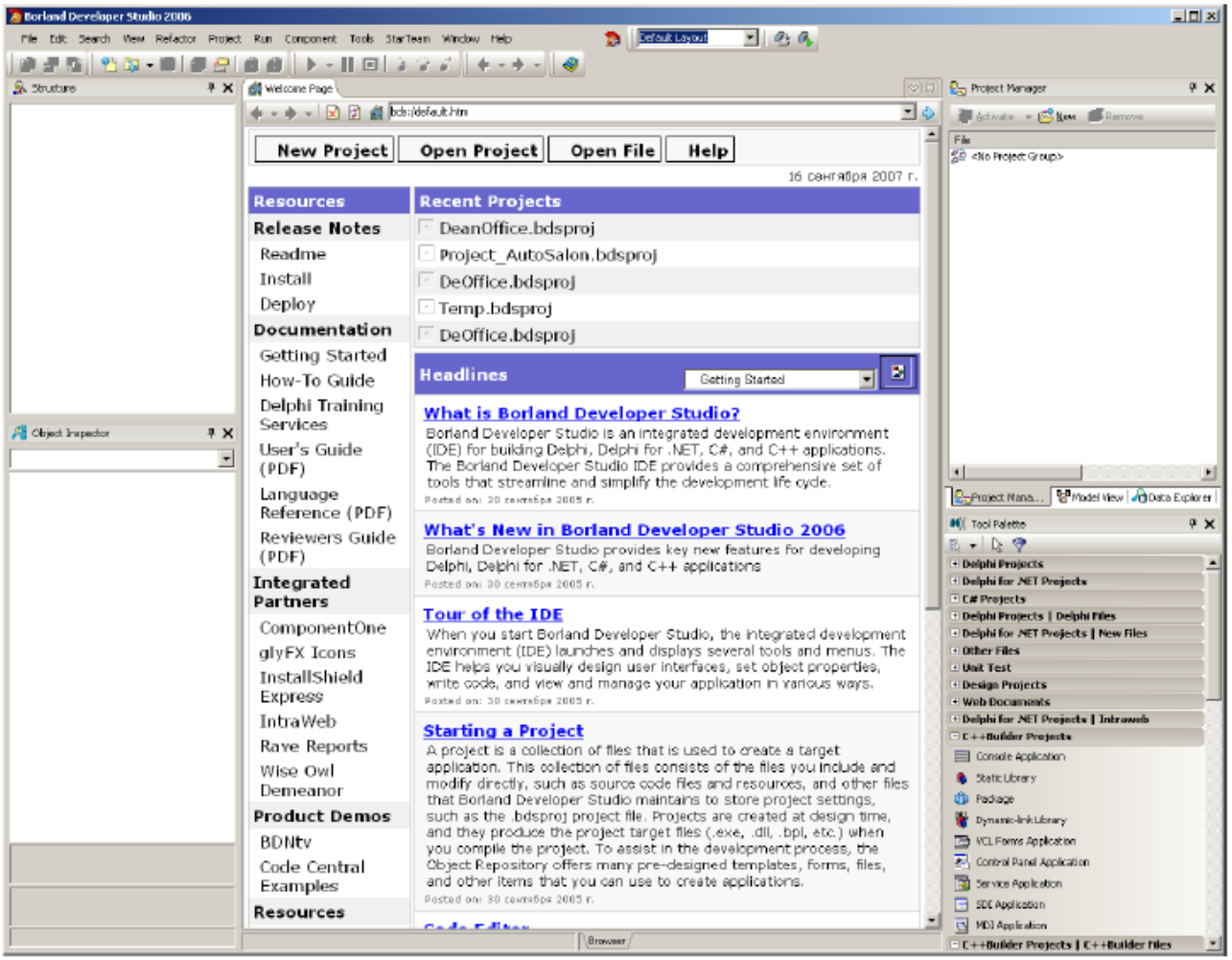


Рисунок 7.1 - Начало работы в среде BDS

2. Создадим пустую заготовку приложения ECO командой File > New > Other. Выберем значок ECO WinForms Application во вкладке Delphi for.NET Projects (рисунок 7.2).

3. В диалоговом окне введем имя проекта (например, projDeanOffice) и его местоположение (каталог). Нажмем кнопку OK.

4. Среда Delphi сформирует пустую заготовку приложения ECO и откроет окно Проектировщика. В нем расположена начальная пустая форма приложения. Структура автоматически созданной модели (пустой заготовки) доступна в окне просмотра модели, открываемом командой View > Model > View либо выбором вкладки Model View в правом верхнем окне (см. рисунок 7.3). В этом окне в дополнение к самому проекту (projDeanOffice) и главной форме (WinForm) можно увидеть еще несколько автоматически добавленных элементов. Среди них имеются следующие:

- элемент projDeanOfficeEcoSpace представляет объектное пространство ECO. Это основное хранилище, в котором располагаются экземпляры классов создаваемой модели во время

работы программы. Это пространство также ответственно за хранение объектов ECO, например, в базе данных или файле XML;

- элемент Package_1 представляет пакет классов UML, которые мы будем создавать.

Переименуем элемент Package_1 в удобное для нас название packModel.

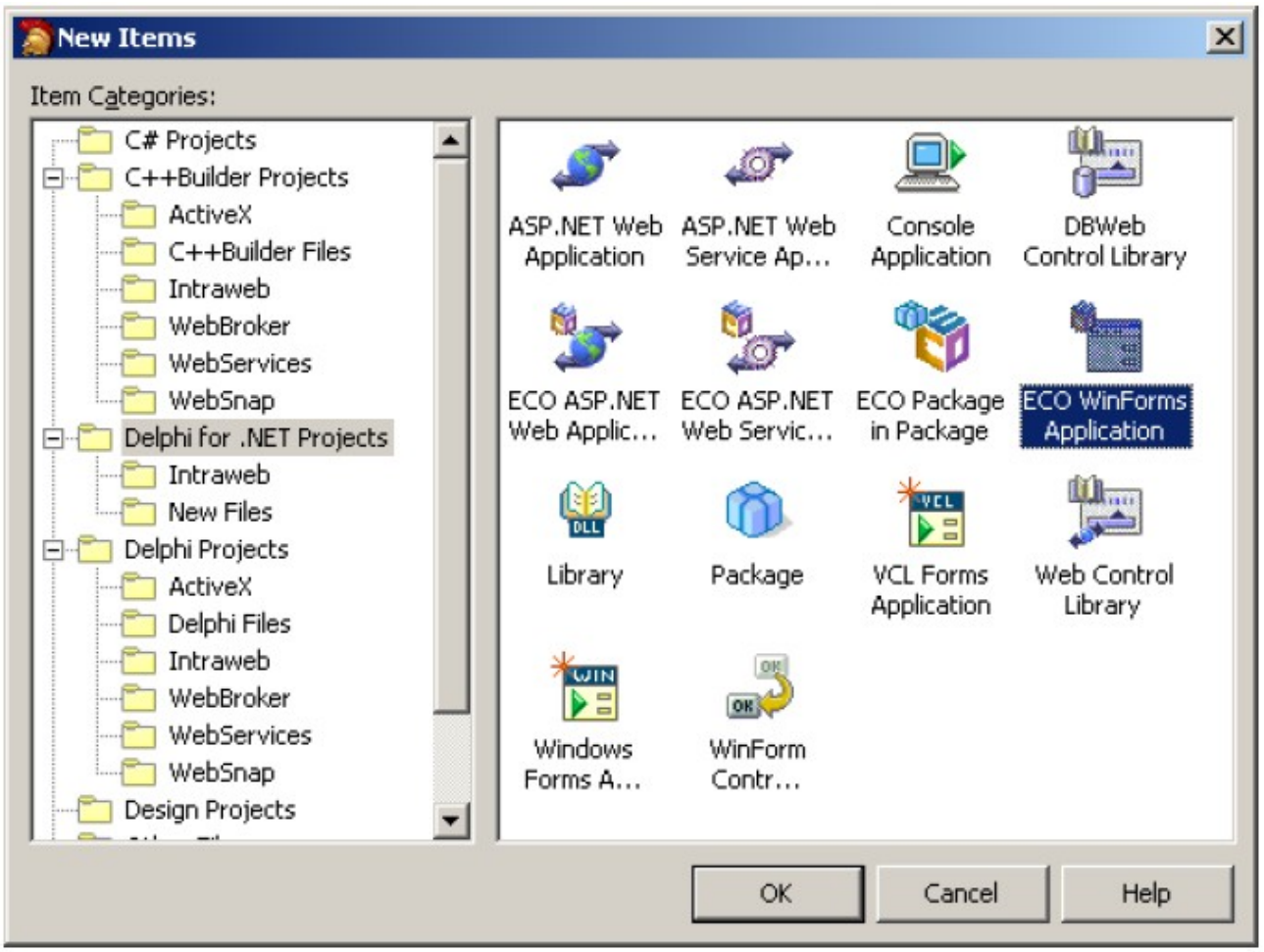


Рисунок 7.2 - Создание заготовки проекта ECO

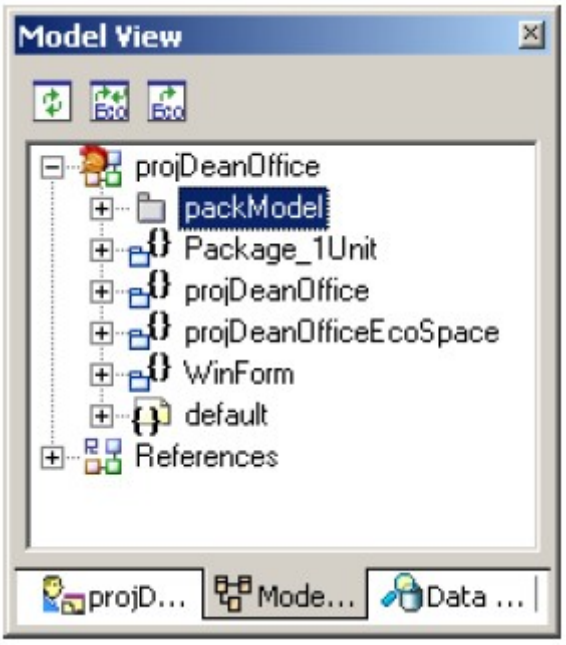


Рисунок 7.3 - Окно просмотра модели

Создание модели UML

1. Для создания модели дважды щелкнем мышью на строке packModel в окне просмотра модели. Откроется окно packModel [diagram] диаграммы классов ECO. Это окно напоминает окно построения обычных диаграмм классов UML, только при работе с ним применяются элементы, специфичные именно для технологии ECO.

2. Разместим на диаграмме первый класс, отражающий сущность Деканат. Для этого выберем на палитре инструментов Tool Palette инструмент ECO Class в категории UML ECO Class Diagram и щелкнем в подходящей точке пространства моделирования. В ней появится графический элемент, изображающий класс.

3. Назовем класс clChair (Кафедра). Пробелы в имени класса ECO не допускаются.

4. Добавим атрибуты класса командой контекстного меню Add > Attribute. Создадим таким образом три атрибута ChairName (Название кафедры), ChairHeadSNP (ФИО заведующего кафедрой), ChairSecrSNP (ФИО секретаря кафедры), рисунок 7.4. Для задания типа атрибута выделим нужный атрибут и в окне Properties установим необходимое значение поля Type в категории General. В данном случае все три атрибута имеют тип String.

5. Переименуем названия класса и атрибутов в понятные названия на русском языке. Для этого в свойстве Alias (категория General) класса и его атрибутов введем русскоязычные названия.

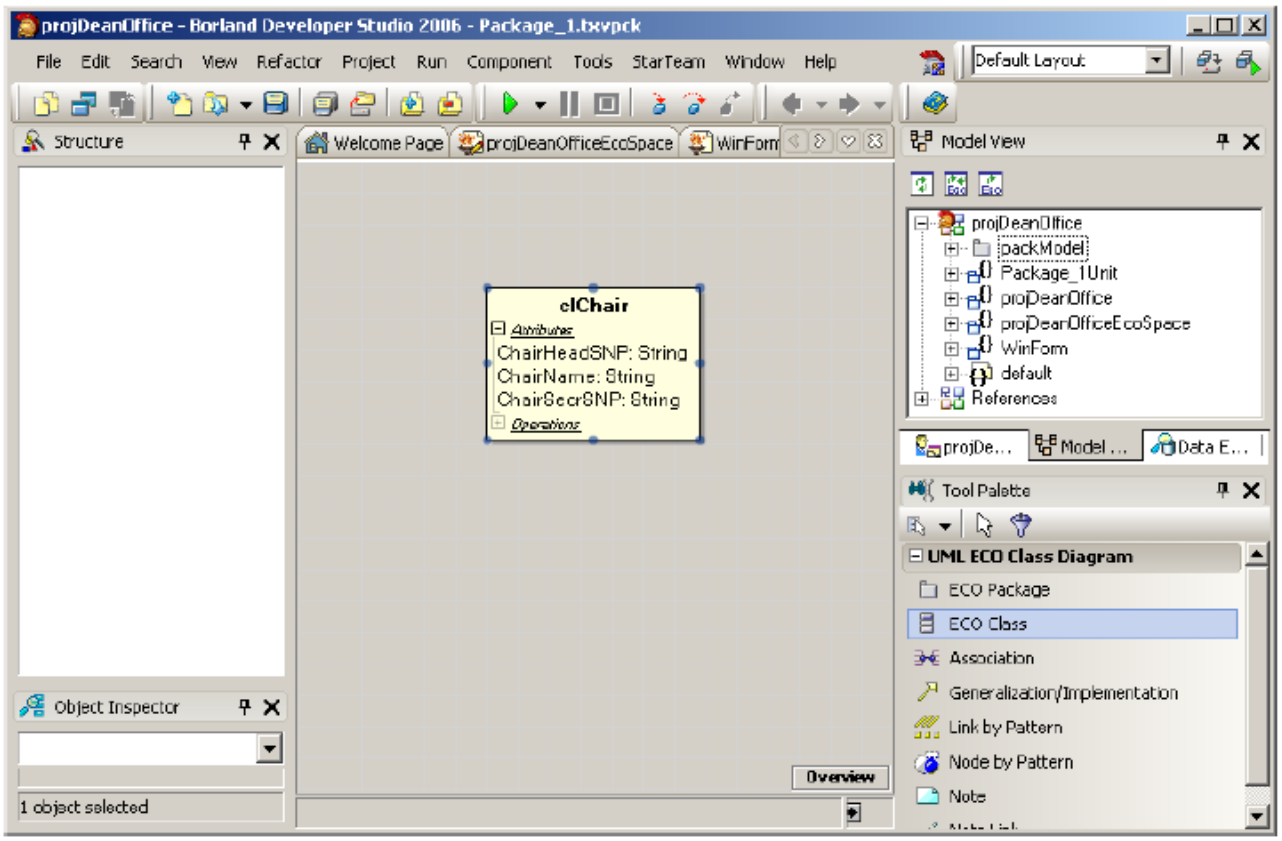


Рисунок 7.4 - Добавление класса clChair с его атрибутами

6. По аналогии добавим к модели еще один класс clLecturer (Преподаватель) с атрибутами LecturerSNP (ФИО преподавателя) и LectAcadDegree (Ученая степень) типа String. Переименуем элементы нового класса на диаграмме, изменив значения свойства Alias.

7. Настроим связи между созданными классами. Эта связь будет представлять отношение ассоциации. Выберем на палитре инструментов инструмент Generalization / Implementation. Щелкнем мышью на представлении класса Кафедра, протянем связь к классу Преподаватель и снова щелкнем мышью. В результате между двумя классами сформируется ассоциативное отношение.

8. Настроим мощность ассоциативного отношения. В нашем случае одному экземпляру класса Кафедра соответствует множество экземпляров класса Преподаватель (от 1 и более). Для этого в окне Properties выделенной связи выберем категорию End1, соответствующую классу Кафедра, и в поле Multiplicity установим значение 1, а в категории End2 (для класса Преподаватель) этому же полю – значение 1..*. Теперь дадим сторонам связи названия. В свойство Name для сторон End1 и End2 введем roleChair и roleLecturers соответственно (рисунок 7.5). Таким

образом, мы назвали роли каждого класса в ассоциативной связи. Эти имена пригодятся нам при кодировании на языке OCL.

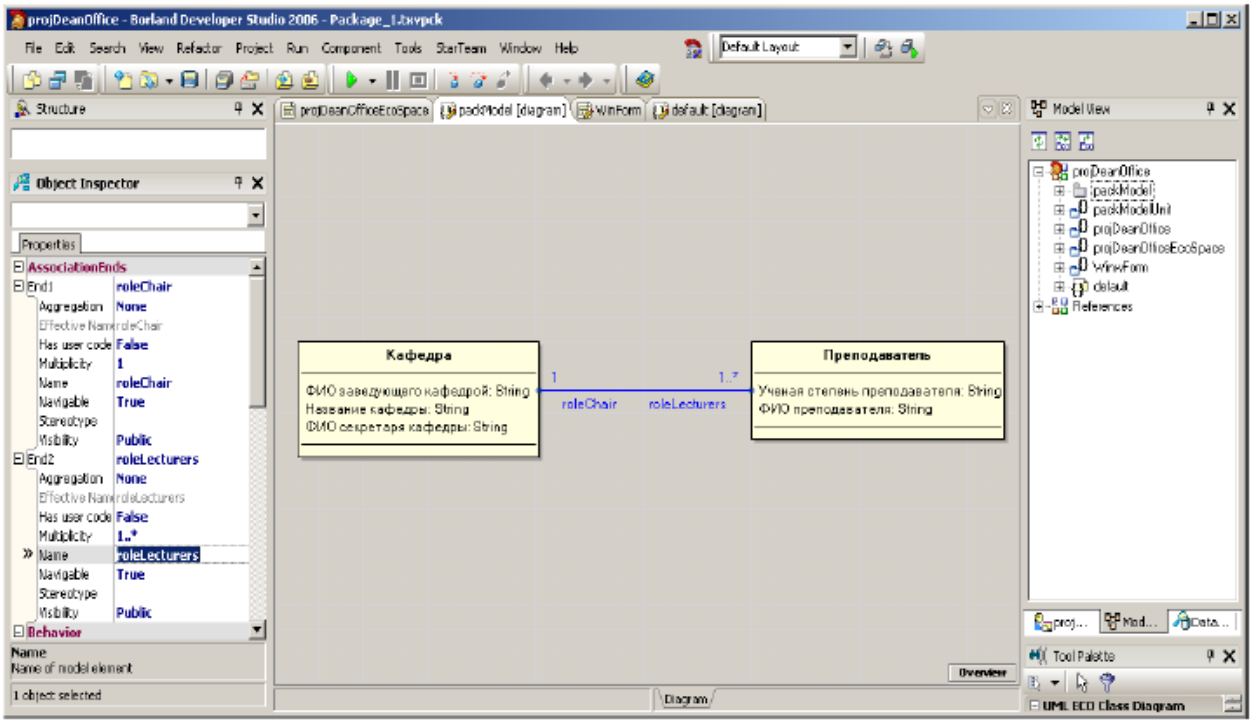


Рисунок 7.5 - Настройка связи между классами

На этом этапе создание простейшей модели закончено. Теперь надо создать базу данных и подключить ее к проекту.

Создание БД и настройка ECO компонент

1. Перейдем на закладку projDeanOfficeEcoSpace рабочего окна Delphi (рисунок 7.6). На этой закладке настраивают особенности функционирования объектного пространства проекта. Пока объектное пространство пусто – оно работает стандартным способом. Элементы модели доступны в других окнах в виде диаграмм UML.

2. Выберем в палитре инструментов компонент BdpConnection из категории Borland Data Provider, предназначенный для связи с СУБД, и добавим его в проект.

3. Настроим этот компонент на доступную базу данных. В качестве учебного примера пропишем путь к пустой базе данных DeOffDB.gbd выбранной СУБД.

4. В свойстве ConnectionString компонента BdpConnection выберем имя соединения IBConn1.

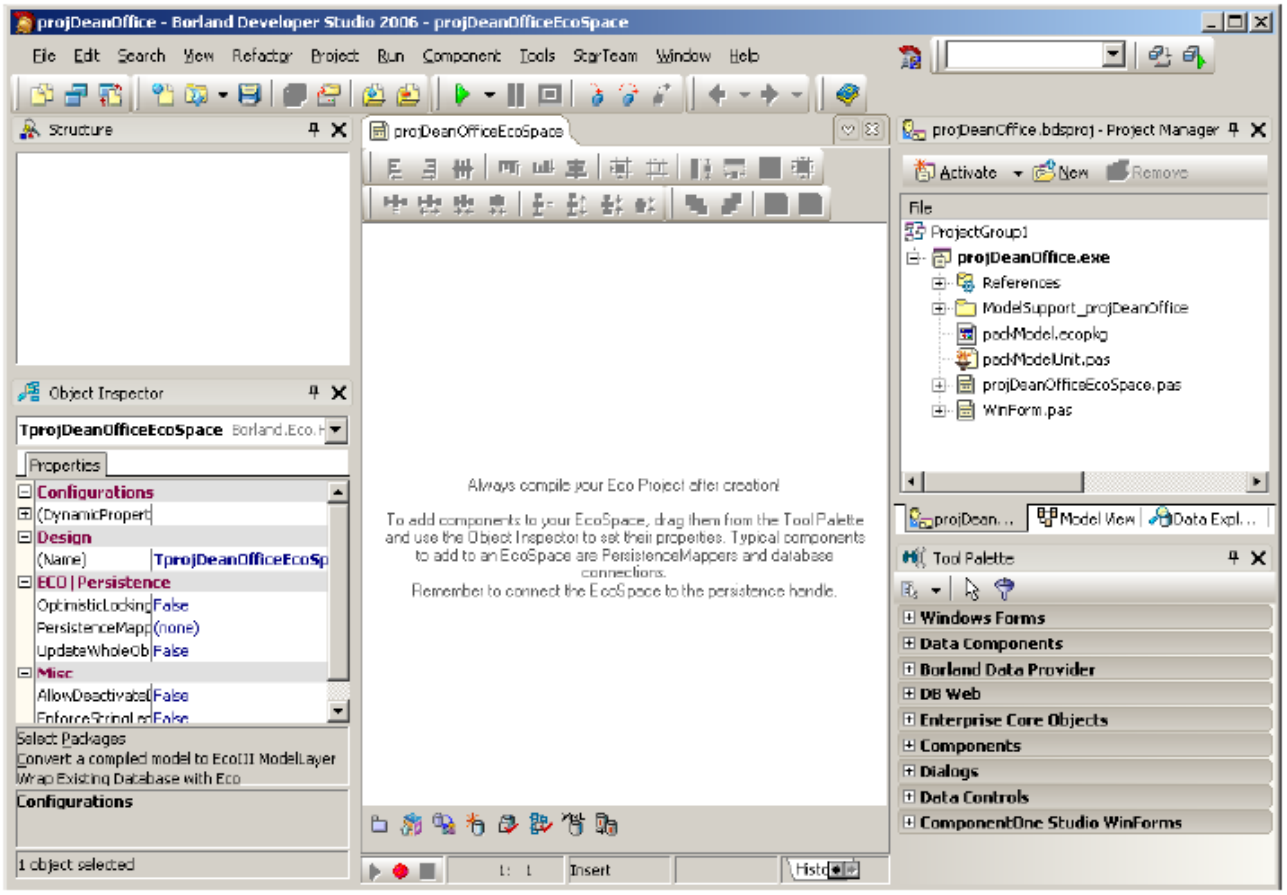


Рисунок 7.6 - Окно настройки объектного пространства

5. Настроим автоматическую связь модели MDA с выбранной СУБД. Синхронизация содержимого пространства ECO с данными на внешних носителях (файлах или базах данных) выполняется компонентами, наследующими базовые характеристики класса PersistenceMapper. Этот класс отвечает за отображение структуры объектного пространства в схемы представления данных, например реляционные. Добавим к проекту компонент PersistenceMapperBdp из категории Enterprise Core Objects. Он предназначен для раскладки объектного пространства в схемы баз данных, доступ к которым происходит по технологии BDP.NET. Этот компонент располагается в окне projDeanOfficeEcoSpace рядом с объектом BdpConnection1 (экземпляром компонента BdpConnection).

6. Свяжем компонент PersistenceMapperBdp через свойство Connection с объектом BdpConnection1 (рисунок 7.7). Компонент PersistenceMapperBdp полностью ответствен за все аспекты автоматического сохранения и загрузки копии объектного пространства модели в выбранную базу данных. Фактически, добавив компонент PersistenceMapperBdp к проекту, мы реализовали все основные аспекты взаимодействия приложения MDA с базой данных.

7. Сгенерируем схему базы данных. Компонент PersistenceMapperBdp самостоятельно создает в выбранной базе данных специальные таблицы, поля и отношения между ними. В этих таблицах хранятся объекты пространства ECO. В контекстном меню компонента PersistenceMapperBdp выберем пункт, обеспечивающий автоматическую настройку всех нужных свойств для конкретной СУБД. В случае использования СУБД Interbase нужный пункт имеет вид Interbase [dialect3] setup, в случае СУБД Microsoft SQL Server – SQL Server setup и так далее. В нашем случае выбираем пункт Interbase [dialect3] setup, так как используется СУБД Firebird 2.0 (рисунок 7.8).

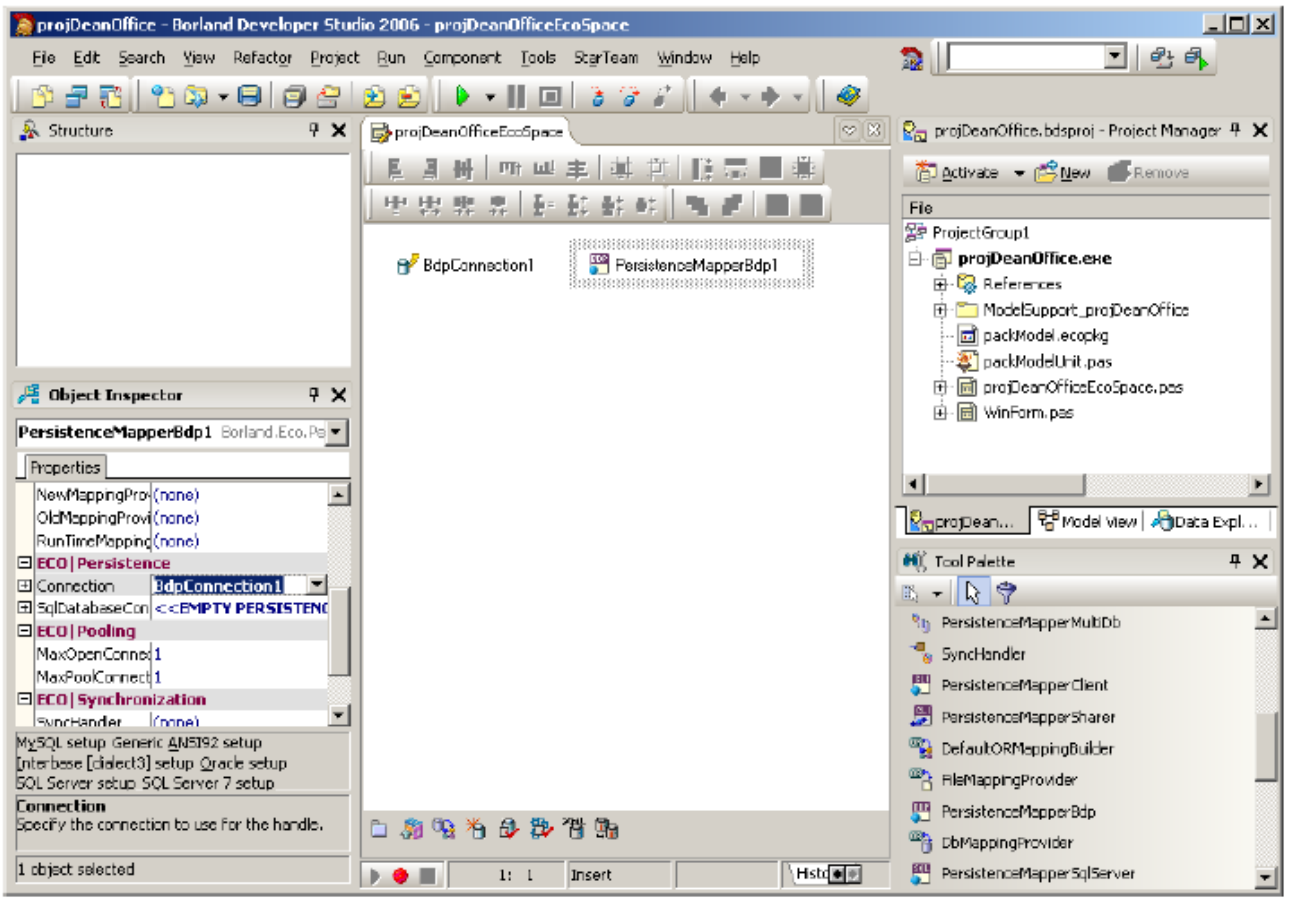


Рисунок 7.7 - Настройка связи ECO с СУБД

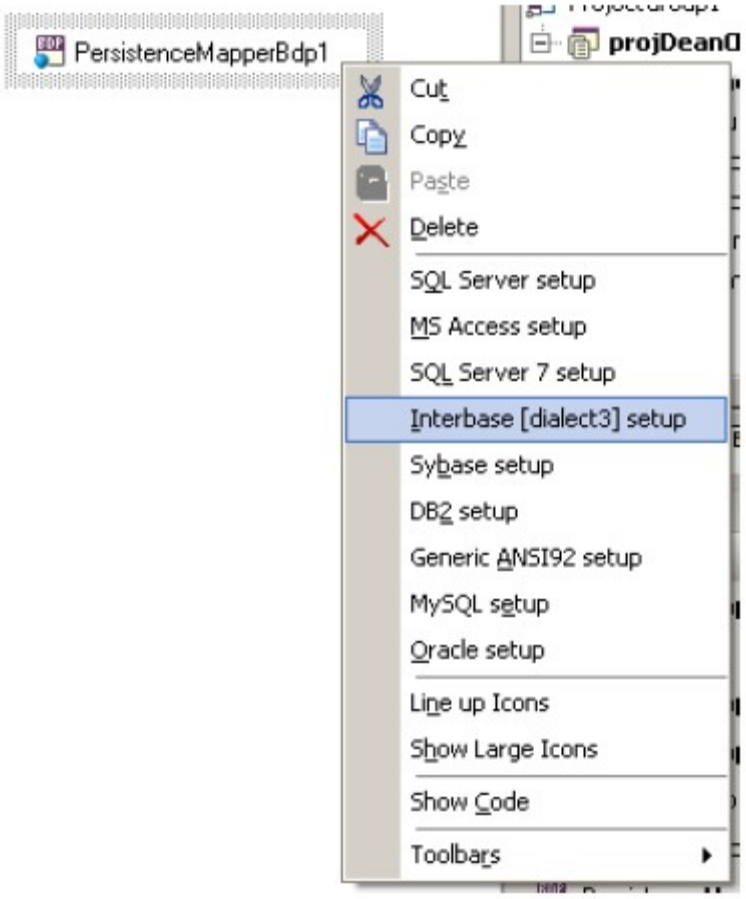
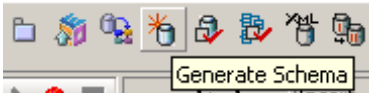


Рисунок 7.8 - Настройка свойств СУБД

8. Сгенерируем таблицы, описывающие структуру объектного пространства. Для этого нажмем кнопку Generate Schema в нижней части окна, представляющего пространство ECO. В текущей базе данных автоматически сформируются таблицы, содержащие поля для хранения модели проекта (рисунок 7.9).



9. В ходе генерации схемы среда Delphi проверит содержимое базы данных. Если в ней имеются таблицы, не имеющие отношения к текущей модели, будет предложено их удалить. По умолчанию режим удаления лишних таблиц выключен.

10. Компонент PersistenceMapperBdp, настроенный на конкретную СУБД, надо задействовать как связующее звено между объектным пространством ECO и СУБД. Для этого обратимся к текущему объектному пространству проекта и щелкнем на свободном месте окна projDeanOfficeEcoSpace.

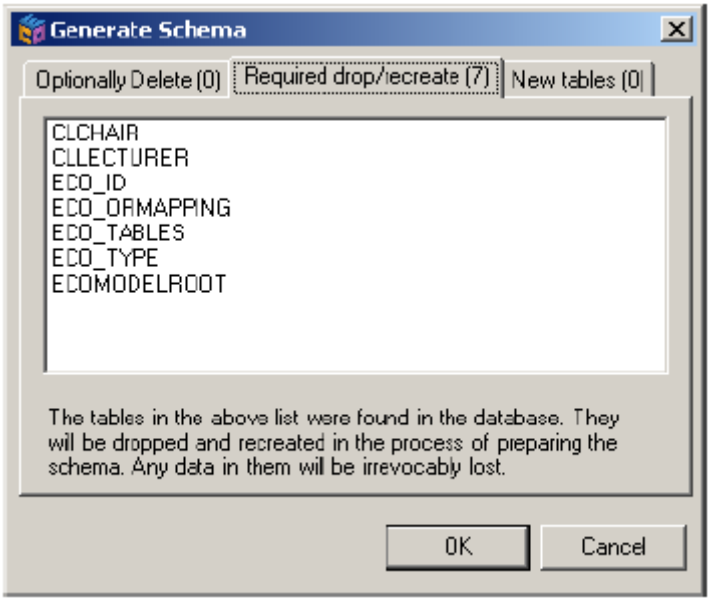


Рисунок 7.9 - Автоматическая генерация схемы модели

В свойстве PersistenceMapper в окне Инспектора объектов Properties для объектного пространства зададим ссылку на объект PersistenceMapperBdp1 (единственный экземпляр компонента PersistenceMapperBdp). Связь объектного пространства с СУБД установлена.

Далее нам необходимо представить создаваемые объекты модели (экземпляры классов Кафедра и Преподаватель) в таблице. Объект следует добавлять в таблицу, удалять его и обновлять БД с помощью кнопок Добавить, Удалить и Сохранить соответственно. Проделаем всю работу исключительно с помощью визуальных средств Delphi, не прибегая к ручному программированию.

Создание интерфейса

1. Перейдем к окну Проектировщика для главной формы проекта WinForm (вкладка Design). Переименуем стандартное название главной формы. Назовем ее wfMain, а сам класс TWinForm переименуем в TMain в свойстве Name категории Design. Свойство Text (категория Appearance) отвечает за название заголовка окна, введем в него строку, например Главная форма. Это окно станет родительским в разрабатываемом приложении, согласно требованиям создания MDI-контейнеров.

2. В свойстве IsMdiContainer (категория Window Style) Главной формы выберем значение True.

3. Создадим новую форму командой File > New > Other. Выберем значок ECO Enabled Windows Form из вкладки New ECO Files (категория Delphi for .NET Projects). В окне Проектировщика появится новая форма. Эта форма будет дочерней, а значит, она будет отображаться внутри главной. Так как в ней будем представлять список преподавателей выбранной кафедры, назовем созданную форму wfLecturer, а сам класс – TLecturer. Введем в свойство формы Text строку Преподаватели.

4. Создадим главное меню на родительской форме для обращения к дочерним формам.

Поместим на Главную форму компонент MainMenu. В верхней части формы появиться строка меню, в начале которой находится область ввода текста. Чтобы создать первый пункт меню, надо щелкнуть в области ввода текста и ввести название пункта меню. Назовем пункт меню Подсистемы. В нижнюю область пункта Подсистемы введем название команды меню – Преподаватели (рисунок 7.10).

5. Выберем форму wfLecturer, вкладку Code. В окне появится код формы. Перед командой Implementation объявим переменную, ответственную за создание дочернего окна Преподаватели:

```
var
callLect: TLecturer;
```

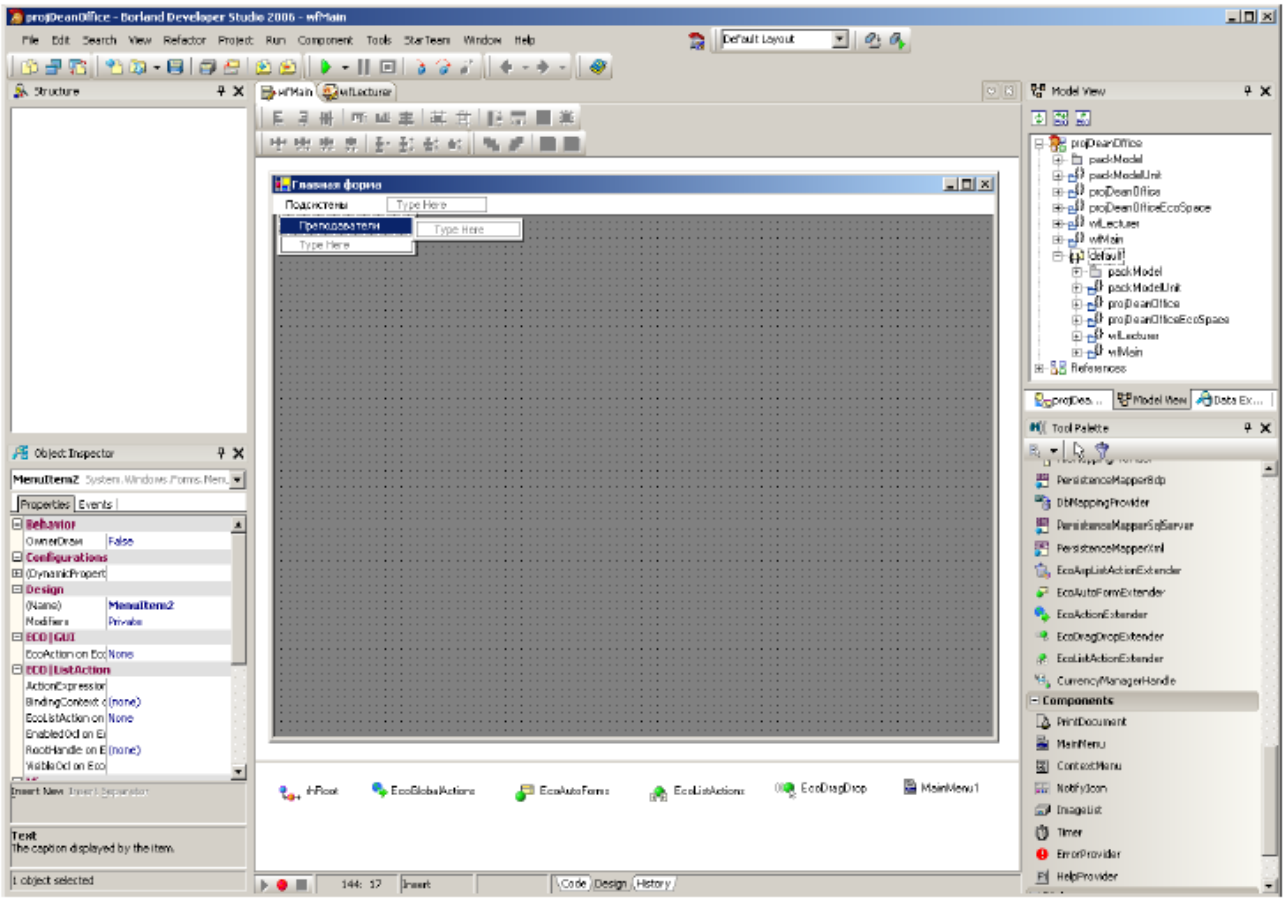


Рисунок 7.10 - Настройка главного меню

6. Перейдем к Главной форме. Чтобы в Главной форме создать дочернюю, подключим к ней форму wfLecturer командой File > Use Unit. В появившемся окне Use Unit выберем форму wfLecturer (Преподаватели). На событие выбора пункта меню Преподаватели пропишем следующие операции:

```
callLect := TLecturer.Create(EcoSpace);
callLect.MdiParent := self;
callLect.Show;
```

7. Запустим приложение. Теперь при нажатии на пункт Преподаватели внутри родительского окна появляется дочернее окно Преподаватели (см. рисунок 7.11).

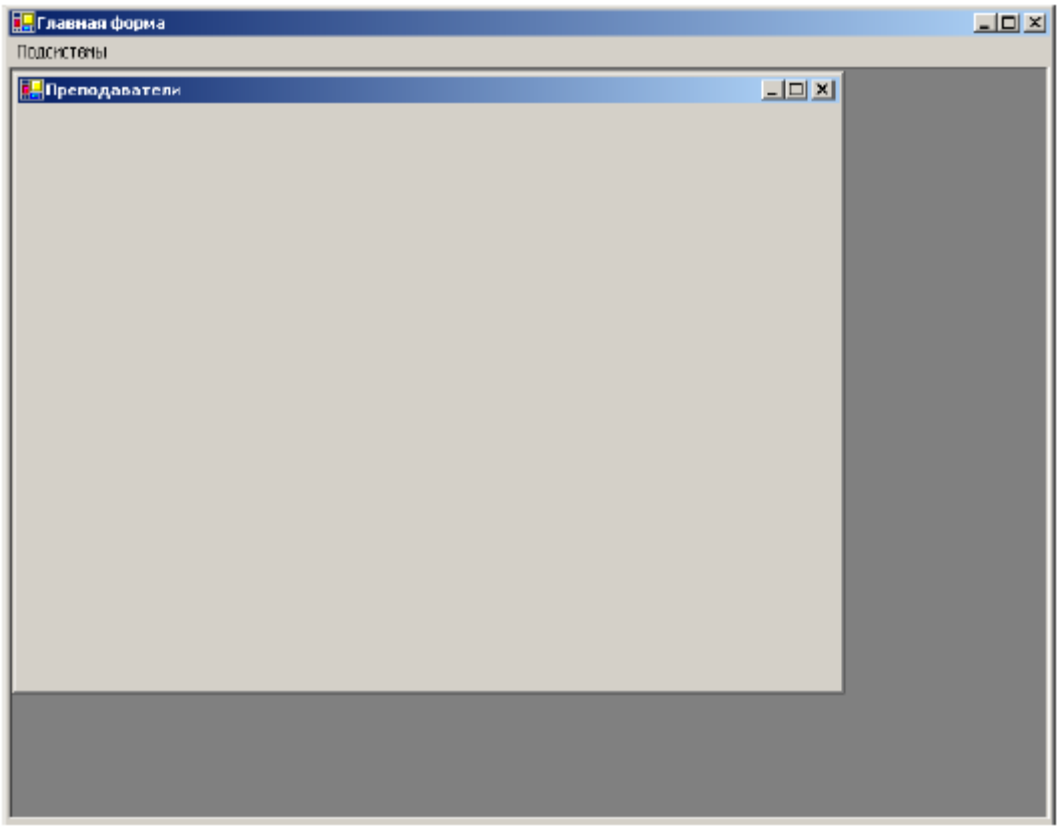


Рисунок 7.11 - Создание дочернего окна внутри родительского

8. Приступим к представлению таблицы с объектами ECO на форме Преподаватели. Воспользуемся готовым компонентом DataGrid из категории палитры инструментов Data Controls. Поместим этот компонент на форму и дадим ему название dgChair (в свойстве Name). Эта таблица будет отвечать за представление экземпляров класса Кафедра (clChair). Исходно таблица должна быть пуста. В свойстве CaptionText (заголовок таблицы) введем название Кафедры.

9. Добавим еще одну таблицу dgLecturer, которая в готовом приложении будет отвечать за отображение экземпляров класса Преподаватель (clLecturer). В свойстве CaptionText введем название Преподаватели.

10. Для работы с таблицами в простом приложении потребуется пять операций: добавление и удаление данных в двух таблицах и сохранение копии ECO пространства из оперативной памяти в базу данных. Редактирование данных будет осуществляться непосредственно в полях таблиц. Добавим на форму пять кнопок – экземпляров класса Button (рисунок 7.12).

11. Настроим автоматическое растягивание таблиц и положение кнопок при изменении размеров окна Преподаватели. Воспользуемся свойством Anchor у объектов формы. Для таблицы Кафедры зададим значения Top, Left, Right в свойстве Anchor, для таблицы Преподаватели – Top, Bottom, Left, Right, для кнопок Button1–Button4 – Top, Right, для кнопки Button5 – Bottom, Right.

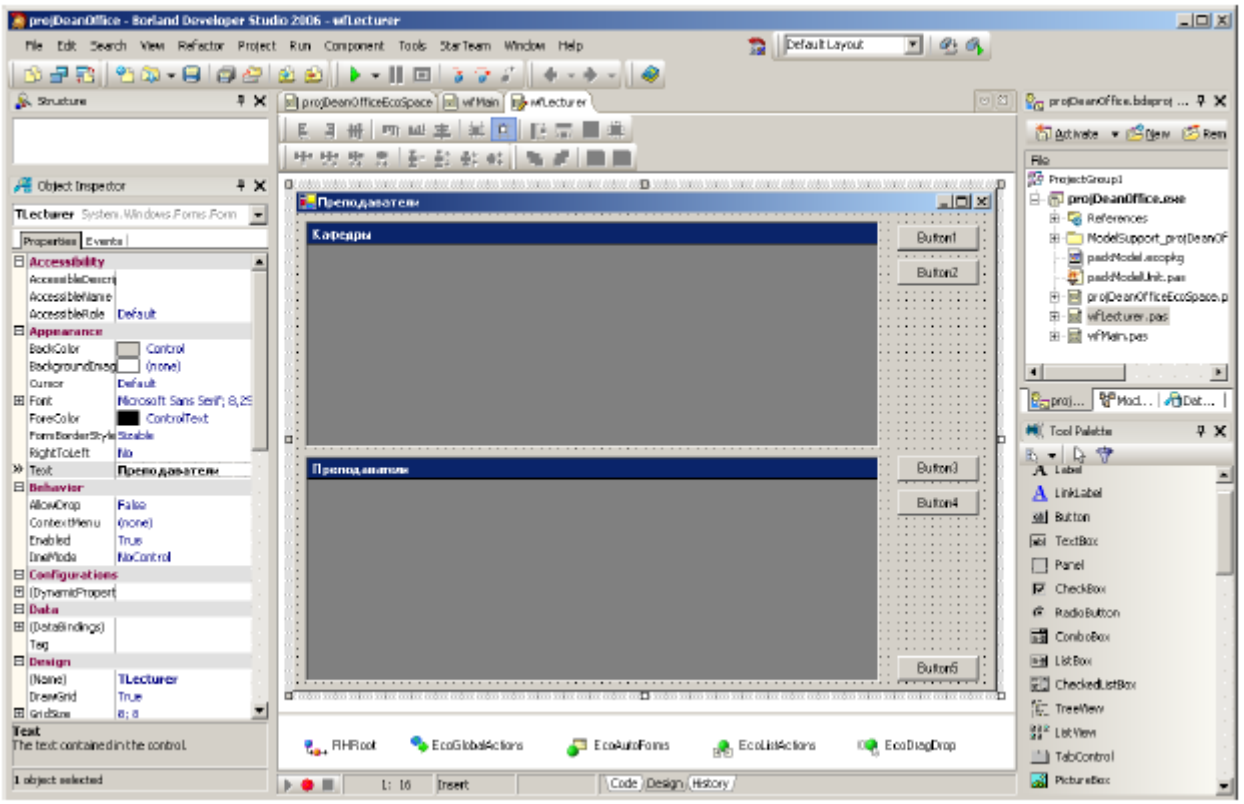


Рисунок 7.12 - Добавление компонентов пользовательского интерфейса

Связывание интерфейса с моделью

1. Связь пользовательского интерфейса с моделями ECO обычно происходит через компонент ExpressionHandle. Он доступен в категории палитры инструментов Enterprise Core Objects. Через подобные идентификаторы (дескрипторы) организуется доступ к объектам модели и пространства ECO во время работы программы.

Идентификаторы ECO связываются друг с другом в цепочки. Во главе такой цепочки расположен корневой идентификатор. Он задает общий контекст работы (доступ к объектному пространству) для всех остальных идентификаторов ECO в программе.

Корневой идентификатор добавляется к проекту автоматически (по умолчанию он называется rhRoot). Все остальные идентификаторы ECO добавляются и настраиваются вручную, что связывает пространство ECO с элементами пользовательского интерфейса с помощью типовых механизмов связывания .NET.

Будем работать с формой Преподаватели. Добавим компонент ExpressionHandle. Он отображается в нижней части окна Проектировщика под формой, где уже имеется набор компонентов ECO. Назовем его ehChair, введем это имя в свойство Name категории Design.

2. В свойстве RootHandle этого дескриптора выберем значение RHRoot – ссылку на родительский, автоматически созданный корневой идентификатор. Так задают место данного идентификатора в цепочке доступа к объектному пространству ECO.

3. В свойстве DataSource таблицы dgChair выберем имя ehChair в списке доступных идентификаторов ECO.

4. Настроим таблицу Преподаватель так, чтобы она отражала список преподавателей, принадлежащих выбранной кафедре в первой таблице. Для начала добавим в проект дескриптор CurrencyManagerHandle (из категории Enterprise Core Objects) и дадим ему имя cmhChair, так как этот компонент будет указывать на текущую строку таблицы Кафедры. Свяжем дескриптор cmhChair с таблицей dgChair через свойство BindingContext. Теперь он отслеживает выделенную строку этой таблицы.

5. Зададим ссылку на объект ehChair в свойстве RootHanler, определяющем корневой идентификатор ECO.

6. Добавим к проекту еще один компонент ExpressionHandle. Назовем его ehLecturer. Дескриптор ehLecturer будет обращаться к экземплярам класса Преподаватель.

7. Зададим ссылку на объект cmhChair в свойстве RootHanler.

8. Потребителем объектов, предоставляемых дескриптором ehLecturer, будет вторая таблица. В ее свойстве DataSource выберем ссылку на объект ehLecturer.

9. Приступим к настройке элементов пользовательского интерфейса. Организуем с помощью визуальных средств Delphi создание новых экземпляров класса Кафедра и добавление их в таблицу. Выделим в окне Проектировщика кнопку Button1. В свойстве EcoListAction (Список стандартных действий ECO) выберем значение Add (Добавить объект).

10. Кнопка Button1 автоматически получила название Add. Переименуем кнопку. Для этого воспользуемся компонентом EcoListActions (он добавляется в проект по умолчанию и находится в нижней части окна Проектировщика), в его свойстве CaptionAdd введем Добавить.

11. Объект ECO, который мы хотим добавить к проекту (сделать доступным через элементы управления на форме), задается в свойстве RootHandle. В нем надо выбрать подходящий поставщик объектов ECO, в нашем случае – идентификатор ehChair.

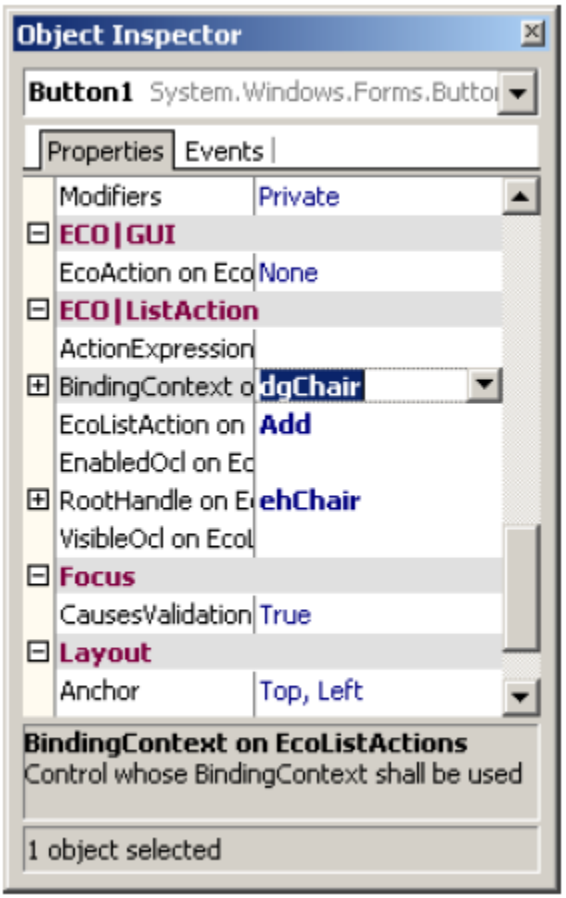


Рисунок 7.13 - Настройка кнопки добавления экземпляра класса Кафедра

12. Свяжем результат действия кнопки (созданный экземпляр класса Кафедра) с визуальным элементом, отображающим этот экземпляр, в нашем случае – с таблицей dgChair. Для этого в свойстве BindingContext (контекст связывания) выберем имя dgChair (см. рисунок 7.13).

13. Теперь добавим операцию удаления экземпляров класса Кафедра. Реализуем это действие по аналогии с операцией добавления. Выделим в окне Проектировщика кнопку Button2. В свойстве EcoListAction выберем значение Delete. Зададим идентификатор ehChair в свойстве RootHandle. Свяжем результат действия кнопки с визуальным элементом – таблицей dgChair. Для этого в свойстве BindingContext у кнопки Delete выберем имя dgChair. Перейдем к компоненту EcoListActions и в его свойстве Caption Delete введем Удалить.

14. Настройка кнопки Button3 (она будет отвечать за добавление экземпляра класса Преподаватель). Значения свойств кнопки: EcoListAction – Add; RootHandle – ehLecturer; BindingContext – dgLecturer. Перейдем к компоненту EcoListActions и в его свойстве Caption Add введем Добавить.

15. Настройка кнопки Button4 (будет отвечать за удаление экземпляра класса Преподаватель). Значения свойств кнопки: EcoListAction – Delete; RootHandle – ehLecturer; BindingContext – dgLecturer. Перейдем к компоненту EcoListActions и в его свойстве Caption Delete введем Удалить.

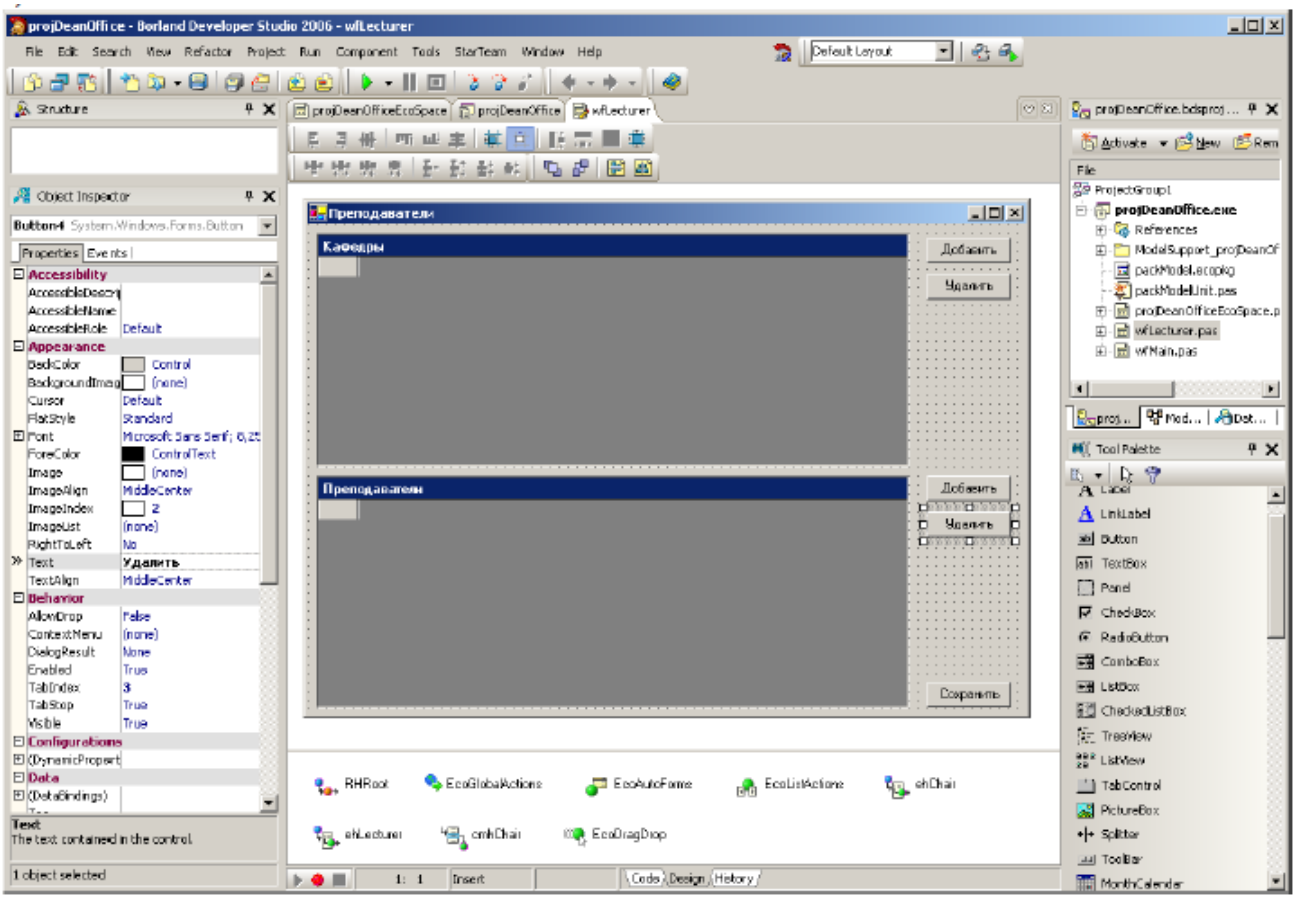


Рисунок 7.14 - Настроенный пользовательский интерфейс приложения

16. Кнопка Button5 будет отвечать за обновление БД. Назовем ее Сохранить. В свойстве EcoAction (категория ECO|GUI) выберем UpdateDatabase. Таким образом, по нажатию этой кнопки выполняется синхронизация содержимого пространства ECO в оперативной памяти и его копии в базе данных. После каждого нажатия кнопки Сохранить содержимое таблицы сохраняется в БД. После нового запуска программы в таблицу пользовательского интерфейса автоматически загружается содержимое модели, сохраненное при последнем выполнении команды UpdateDatabase.

На данном этапе связывание интерфейса с моделью закончено (см. рисунок 7.14).

Создание логики на OCL

1) Действия, которые выполняет дескриптор в программе, определяются значением свойства Expression. В это свойство вводится выражение языка OCL, которое определяет схему создания объектов модели ECO.

Используем дескриптор ehChair для обращения к экземплярам класса Кафедра. Введем в свойство Expression объекта ehChair строку clChair.allinstances. Иначе это можно сделать с помощью редактора OCL - выражений. Введенное выражение задает доступ ко всем текущим экземплярам класса Кафедра в объектном пространстве. В таблице, показанной на форме в окне Проектировщика, сразу же отобразятся поля класса Кафедра (ChairHeadSNP, ChairSecrSNP и ChairName) (в том числе поля родительского класса, если он имеется). Дескриптор ehChair становится поставщиком данных нашей модели ECO для первой таблицы.

2. В качестве выражения OCL объекта ehLecturer введем строку self.roleLecturers в свойство Expression. Здесь roleLecturers – это введенное нами при построении модели имя роли класса Преподаватель (clLecturer) в его ассоциативной связи с классом Кафедра (clChair). Это выражение определяет группу преподавателей, принадлежащих кафедре, которая выбрана в первой таблице. Дескриптор ehLecturer становится поставщиком данных для второй таблицы (см. рисунок 7.15).

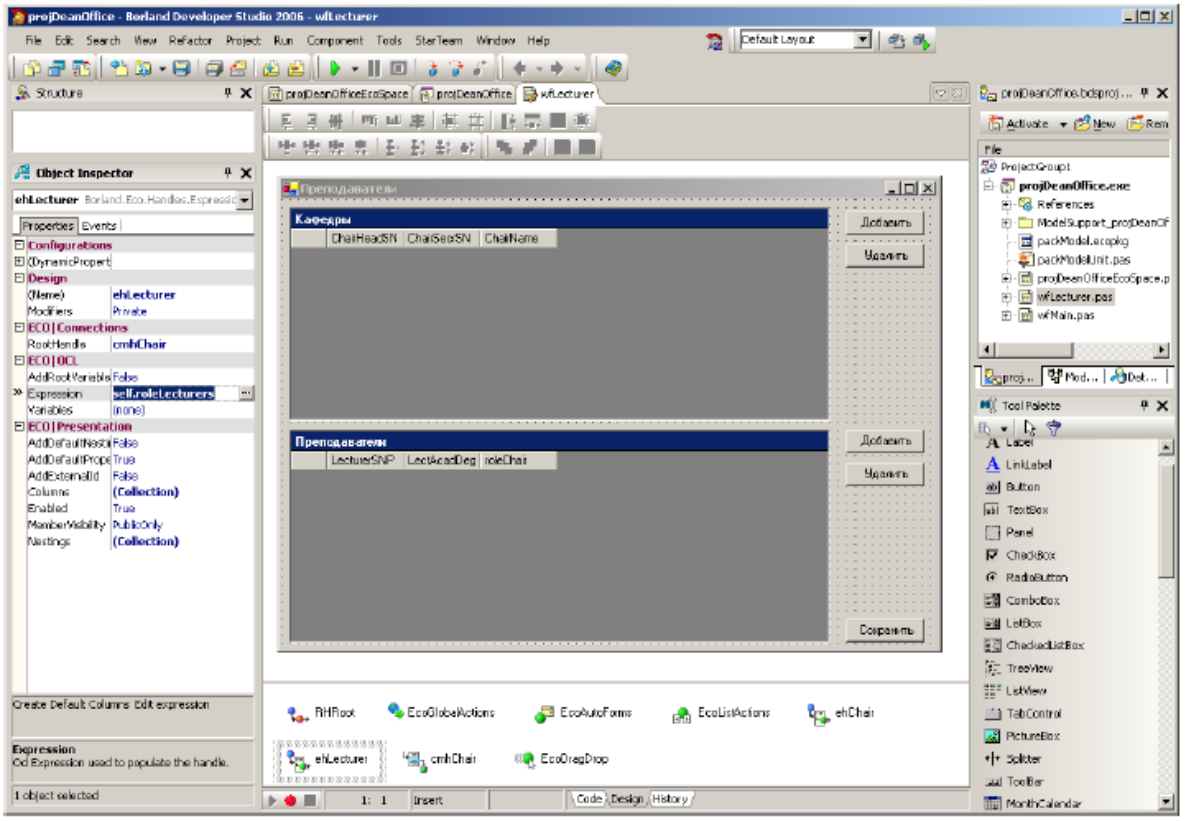


Рисунок 7.15 - Настройка OCL-выражений для взаимосвязанных таблиц

3. Дадим заголовкам полей таблиц русскоязычные названия. Выберем на форме таблицу dgChair. Щелчком на кнопке с многоточием в строке свойства TableStyles откроем окно редактора коллекции стилей таблицы DataGridTableStyle Collection Editor. Создадим новый стиль таблицы, нажав на кнопку Add (см. рисунок 7.16).

Щелкнем по кнопке с многоточием в строке свойства GridColumnStyles. Откроется окно редактора стиля поля таблицы. Нажмем кнопку Add 3 раза, так как нужно переименовать три поля (создать три стиля для имеющихся полей таблицы). Выберем свойство HeaderText первого элемента коллекции и введем в него стоку Название кафедры. Затем в свойстве Width установим ширину поля – 150. И в свойстве MapingName выберем ChairName, т. е. первый столбец настраиваемой таблицы будет содержать информацию атрибута ChairName класса Кафедра (см. рисунок 7.17).

Теперь настроим оставшиеся два элемента коллекции. Эти столбцы будут называться ФИО заведующего кафедрой и ФИО секретаря кафедры. Нажмем кнопку ОК в обоих открытых окнах и убедимся, что поля таблицы получили новые названия.

4. По аналогии настроим стиль таблицы Преподаватели. В ней отобразим два поля: LecturerSNP и LectAcadDegree. Назовем их ФИО преподавателя и Ученая степень.

5. Запустим программу и убедимся, что при нажатии кнопки Добавить в таблице автоматически формируются новые строки, отражающие содержимое новых экземпляров класса из объектного пространства ECO. В них можно ввести нужные значения. Проверим корректное функционирование остальных настроенных операций над объектным пространством (см. рисунок 7.18).

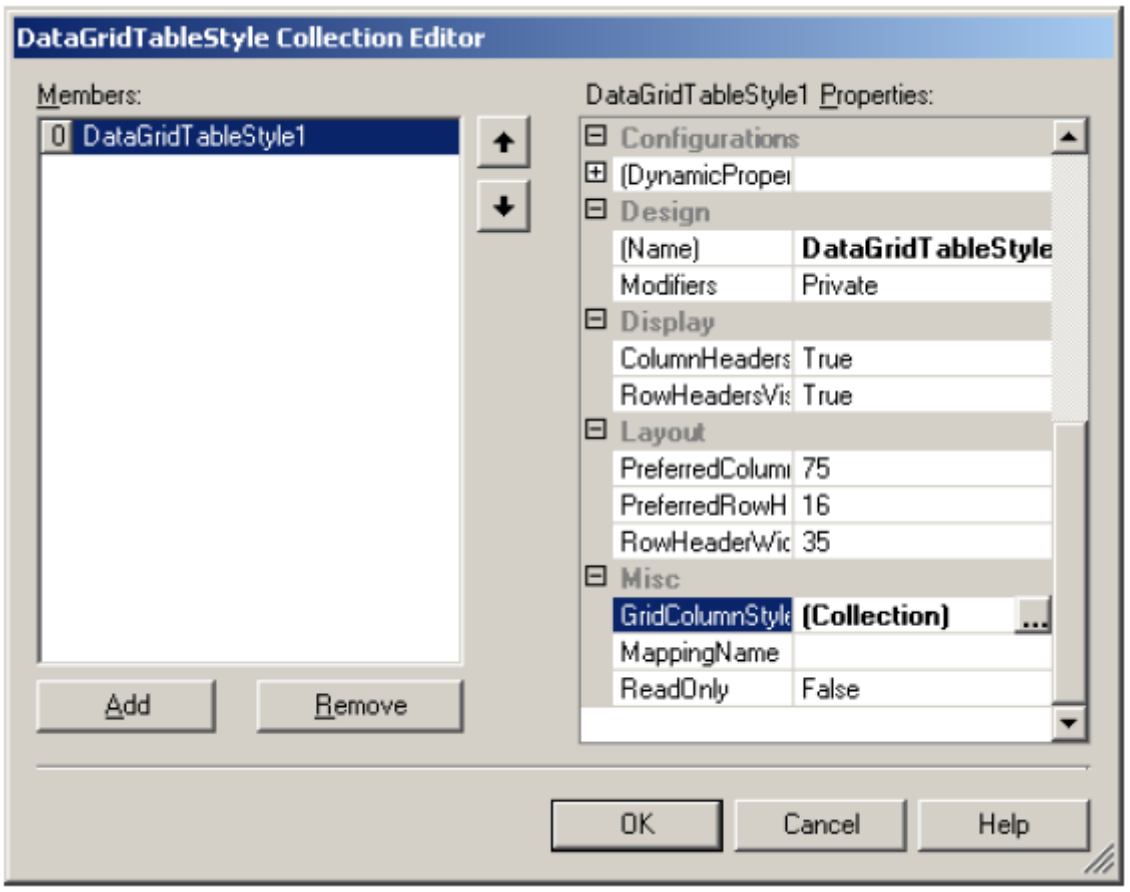


Рисунок 7.16 - Коллекция таблиц компонента gdChair

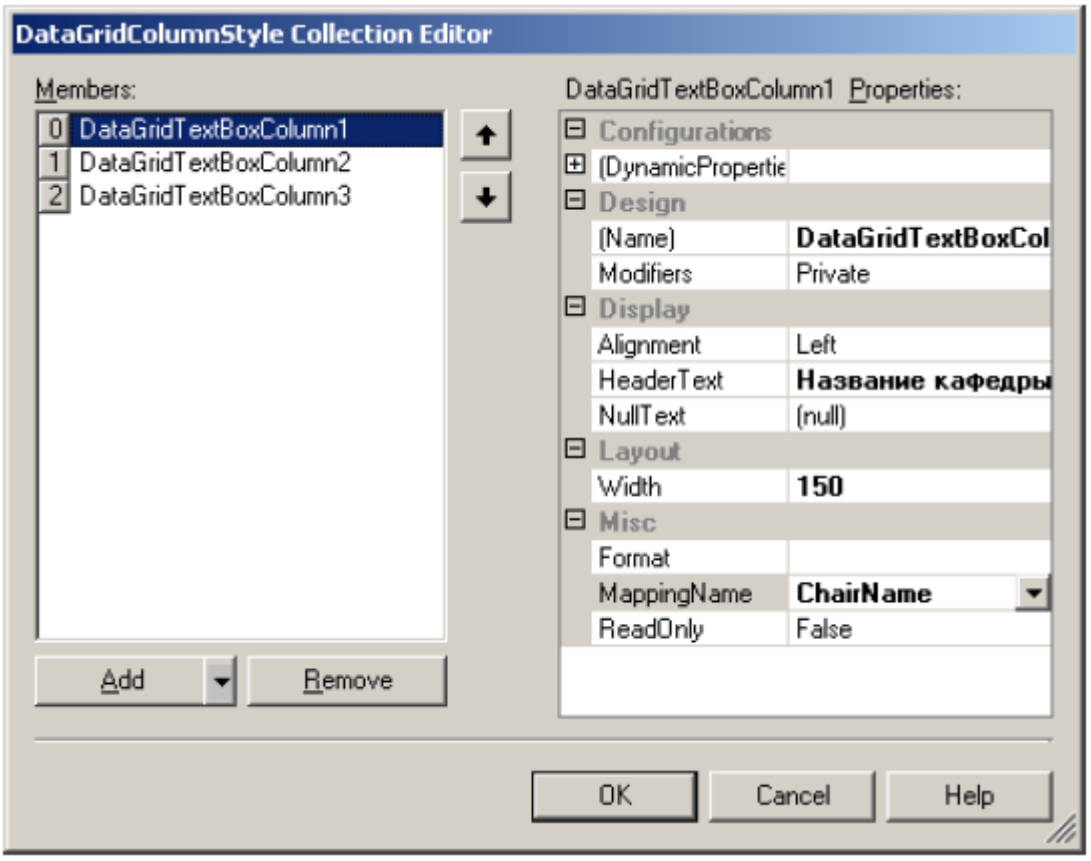


Рисунок 7.17 - Настройка элемента коллекции столбцов

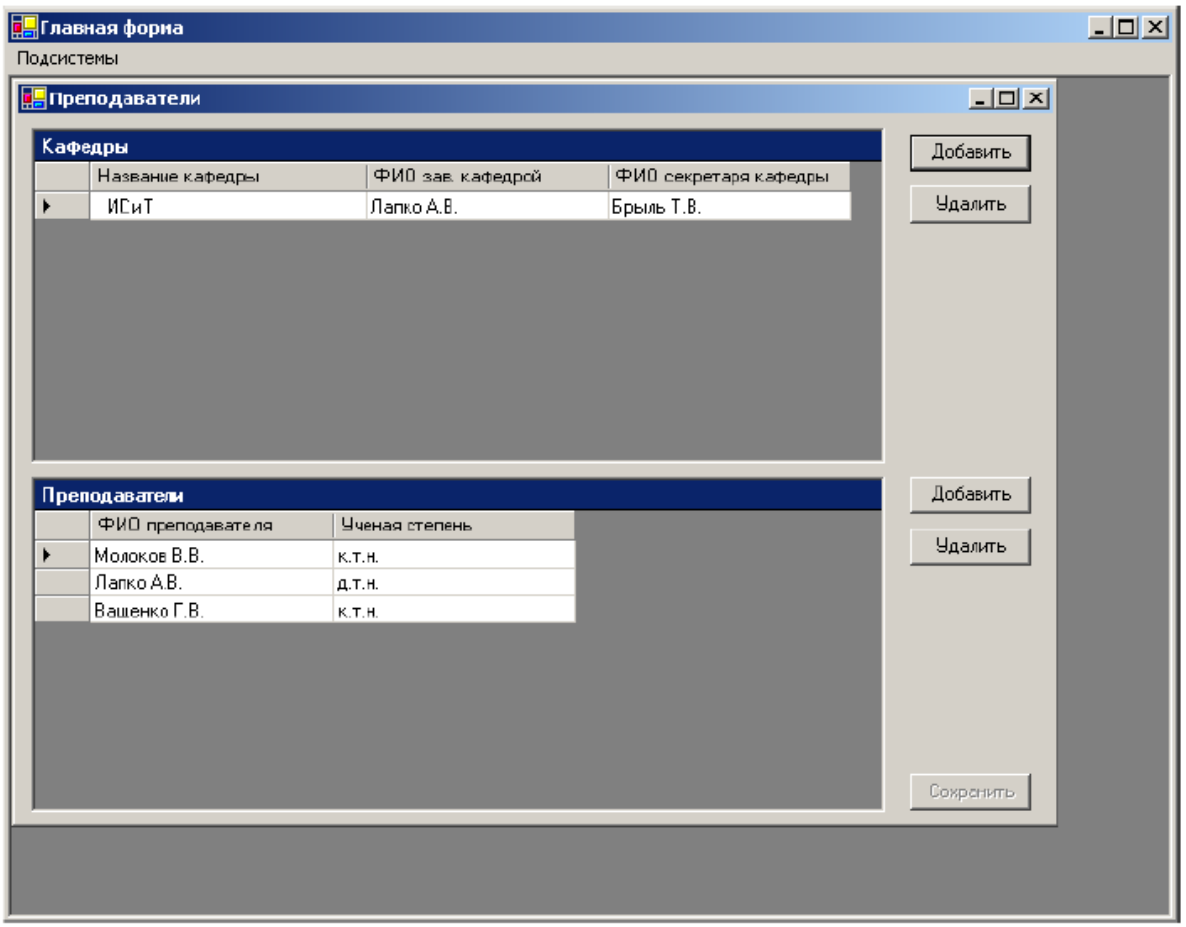


Рисунок 7.18 - Работа с базой данных в режиме таблицы

Задания и порядок выполнения работы

1. Сформировать модель UML. Создать классы, описать их атрибуты. Настроить взаимосвязи между классами.
2. Связать пространство ECO с базой данных.
3. Создать пользовательский интерфейс.
4. Связать интерфейс с моделью UML.
5. Элементы управления связать с выражениями OCL для выполнения типичных стандартных действий (добавление, удаление объектов ECO и сохранение их в базу данных).

Контрольные вопросы

1. Что представляет собой технология MDA?
2. Что такое технология ECO?
3. Зачем нужны дескрипторы ECO?
4. Зачем нужен язык OCL?
5. Из каких этапов складывается процесс построения приложения ECO?
6. Как создаются классы в рамках модели приложения ECO?
7. Как представители класса PersistenceMapper настраиваются на конкретные СУБД?
8. Как автоматически сгенерировать схему базы данных на основе модели ECO?
9. Как элементы пользовательского интерфейса связываются с моделью ECO?
10. Как работает визуальный редактор выражений OCL?

Работа с литературой:

Рекомендуемые источники информации (№ источника)			
Основная	Дополнительная	Методическая	Интернет-ресурсы
1-3	1-3	1-3	1-2

Оценочные средства: контрольные вопросы

Тема 5. Технология MDA.

Лабораторная работа №8. Разработка MDA-приложения с использованием машин состояний

Форма проведения: Практическое выполнение задания
Цель работы – научиться использовать машины состояний при создании MDA-приложений.
Краткие теоретические сведения

Автоматы

Работа объектов программы описывается с помощью *автоматов* (или *машин состояний).* Существует хорошо развитая *теория автоматов,* в которой изучаются вопросы их построения и анализа работы.

Автомат – это элемент системы, который характеризуется двумя базовыми понятиями: состояние и переход. Каждый автомат в определенный момент времени может находиться в одном из допустимых *состояний.* Число этих состояний, возможно, бесконечно. Задан также набор правил, по которому допускается *переход* из одних множеств состояний в другие.

Схожесть автоматной модели с программированием в том, что переменные (ячейки памяти) в каждый момент работы процессора всегда хранят конкретные значения из допустимых диапазонов значений. Эти значения фактически являются состояниями переменных.

Разрешенные над значениями переменных вычислительные операции приводят к переходу этих переменных в новые состояния (в новые значения).

Этот алгоритмический принцип может быть распространен на большое число управленческих задач. В общем случае автоматный подход позволяет описать любую, сколь угодно сложную последовательность операций.

Познакомимся с *базовыми правилами работы автоматов в языке UML*.

Автомат в определенный момент времени может находиться только в одном состоянии. Это правило соответствует принципу императивного программирования, согласно которому любая переменная в каждый момент времени (такт процессора, шаг работы программы) хранит только одно из множества допустимых значений.

Число состояний автомата конечно.

Из текущего состояния автомат может перейти только в одно следующее состояние. Не исключено, что это состояние выбирается из нескольких доступных вариантов.

В рамках диаграммы у каждого состояния должен быть предшественник (за исключением начального состояния). Это правило запрещает размещение на диаграмме состояний, никак не связанных друг с другом.

В определенном состоянии автомат может находиться сколь угодно долго. Переходы между состояниями считаются мгновенными. На диаграмме состояний не регистрируется путь (история состояний), приведший автомат в текущее состояние (в отличие от диаграмм последовательности и кооперации). Это правило показывает, что по текущему значению переменной нельзя определить те значения (состояния), в которых она находилась ранее.

Состояния

Конкретное состояние (в общем случае – статический срез системы) формируется инструментом State (Состояние). На диаграмме оно изображается в виде прямоугольника с закругленными углами. Состояние имеет *имя,* начинающееся с заглавной буквы. Под ним могут записываться *условия* и *действия.* Условия проверяются, а действия выполняются, когда автомат находится в соответствующем состоянии.

Отметим принятые правила оформления элементов диаграммы состояний. Элемент State представляет не действие, а описание статического состояния, в котором автомат может находиться неограниченно долго. Автомат может переходить из текущего состояния в другие состояния при выполнении определенных действий пользователя или условий, которые должны быть выполнены. Действия и условия отображаются только на линиях переходов между состояниями.

Подавтоматы

Внутри каждого состояния разрешается формировать собственный автомат (подавтомат) – мини-диаграмму состояний. Благодаря этому возможно проектировать систему постепенно, раскрывая отдельные ее элементы по мере необходимости.

Подавтомат – это автомат, вложенный в другой автомат и описывающий поведение конкретного состояния.

Диаграммы состояний

Диаграммы состояний (State Machine Diagram) стали новым типом диаграмм в версии UML

2.0. Они особенно важны для разработчика, использующего среду Delphi. Дело в том, что в версии Delphi 2006 имеется технология моделирования ECO III. Она расширена средствами визуального построения алгоритмов. С помощью этих средств описывается работа разных элементов модели. Ранее для описания модели и генерации исходного кода на языке Delphi применялись лишь статические диаграммы классов. Теперь задействованы и диаграммы состояний.

Диаграмма состояний – это средство, описывающее логику функционирования автоматов (машин состояний).

В основу диаграмм состояний положена концепция автоматов. Доказано, что с помощью таких автоматов можно запрограммировать произвольный алгоритм любой сложности, если его можно также записать на императивном языке программирования.

Диаграмма состояний всегда относится к конкретному классу и описывает его внутреннее функционирование. На диаграмме конкретное состояние отображается с помощью элемента State. Начальное и конечное состояния – элементы Initial (Начальное) и Final (Конечное) – представлены на диаграмме сплошным кружком. Кружок, соответствующий конечному состоянию, обведен каймой. Фактически, это псевдосостояния, не возникающие при реальной работе. Начальное и конечное состояния лишь наглядно задают последовательность входа в первое рабочее состояние и выхода из последнего рабочего состояния.

Отдельное состояние может охватывать последовательность действий. Состояние, охватывающее другие состояния, называют *суперсостоянием,* а вложенные в него состояния – *подсостояниями.* Такой иерархический подход к организации состояний позволяет формировать одинаковые реакции подсостояний на уровне одного суперсостояния. Пусть, например, имеется стандартное аварийное состояние и стандартный переход в него по команде отмены. Для каждого из множества состояний диаграммы можно указать этот переход индивидуально, а можно объединить их в суперсостояние. Тогда переход в аварийное состояние представляется на диаграмме всего одной линией, исходящей из элемента, представляющего суперсостояние.

Каждое из состояний на диаграмме не обязано пассивно ожидать внешнего события, приводящего к переходу основного объекта в следующее состояние. Состояние может вести определенную деятельность. Соответствующая деятельность задается свойством состояния Do activity (Выполнение деятельности). Нужная деятельность уже должна быть задана в проекте, например на одной из диаграмм деятельности. Подобное состояние, когда до него доходит очередь, начинает эту деятельность и ожидает ее завершения, фактически отображая не статическое состояние, а выполнение процесса. Этот процесс может быть прерван, что вызовет переход в другое состояние. Если он закончится успешно, также выполнится соответствующий переход.

В диаграммах состояний UML 2.0 можно описывать историческое состояние. Обычная история состояния и глубокая история представлены как отдельные элементы: Shallow History (Обычная история состояния) и Deep History (Глубокая история).

Элемент Junction (Слияние) визуализирует слияние и разделение потоков управления. Он представляет собой промежуточный узел, по внешнему виду аналогичный начальному элементу Initial. В таком узле собираются, а потом вновь расходятся потоки управления. В диаграммах состояний можно также задействовать условный элемент Choice (Выбор).

Создание MDA-приложений с использованием машин состояний

Для примера возьмем уже созданное нами MDA-приложение в лабораторной работе № 7 и дополним его машиной состояний. Расширим нашу предметную область. Добавим новый класс Дисциплина. Теперь у преподавателей появится список преподаваемых ими дисциплин. Введем ограничение на количество рабочих часов в семестр для преподавателя. Задача: запрограммировать процесс распределения нагрузки на преподавателей.

Использование диаграмм состояний UML позволяют визуально проектировать последовательности смены состояний объектов программы в зависимости от условий.

Модификация модели UML

1. Откроем проект projDeanOffice.

2. Добавим к модельному пространству приложения класс clSubject (Дисциплина). Определим поля класса: поле SubjName (Название дисциплины) типа String, поле SubjType (Тип дисциплины) типа String, поле SubjAmountHours (Количество часов дисциплины) типа Integer.

3. В существующий класс Преподаватель добавим атрибут MaxAmountHours (Ограничение на количество рабочих часов) типа Integer.

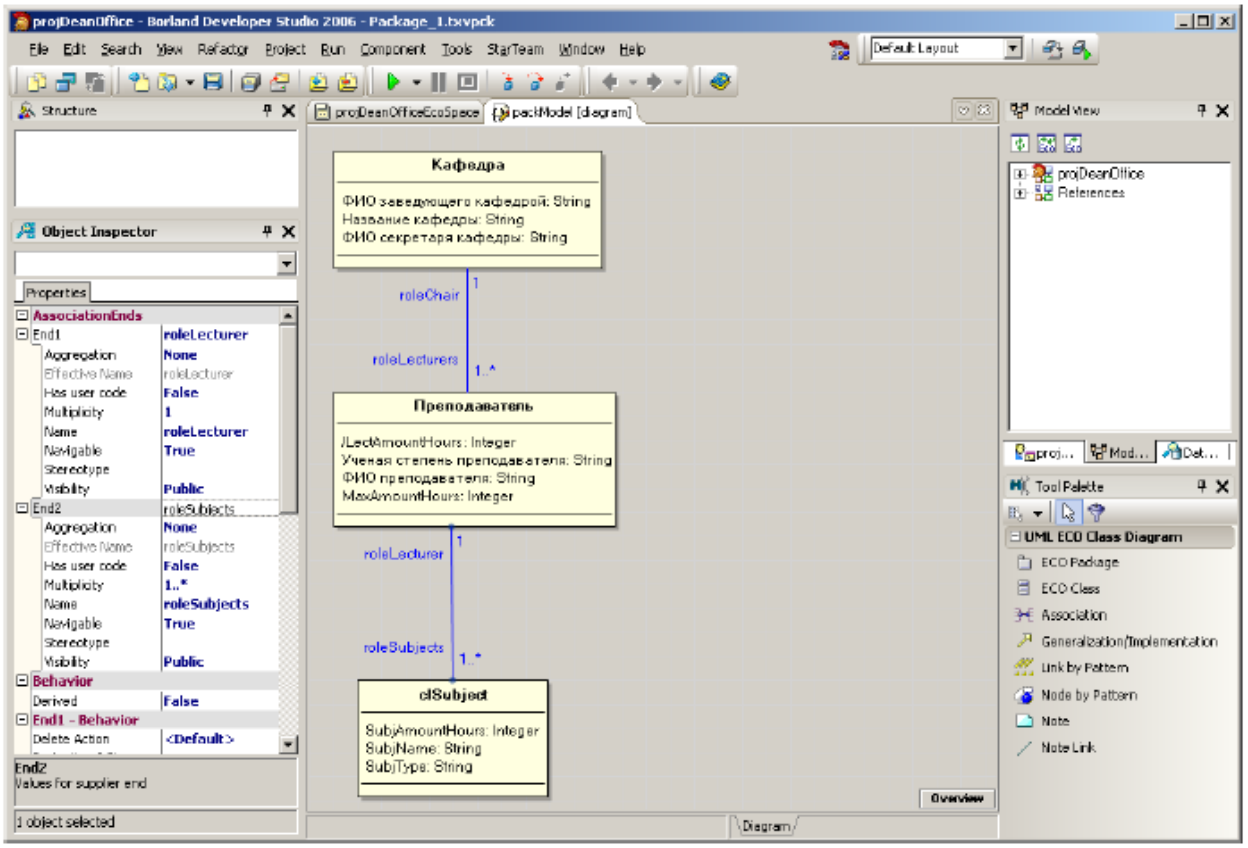


Рисунок 8.1 - Модификация модели UML

4. Теперь добавим вычисляемое поле. Вычисляемое поле – это поле, значение которого не хранится в классе, а рассчитывается непосредственно в момент обращения к нему. Добавим в класс Преподаватель атрибут LectAmountHours (Количество рабочих часов в семестр) типа Integer. Перед именем поля поставим символ /, визуально обозначающий вычисляемое значение. Признаком вычисляемости поля является значение True для свойства Derived (вычисляемое). Значение свойства LectAmountHours будет формироваться выражение OCL. Нужное выражение должно быть значением свойства Derivation OCL (Код OCL для вычисления). Это выражение введем позже, когда создадим машину состояний.

5. Сформируем между созданными классами ассоциативное отношение: «один преподаватель – много дисциплин». Дадим сторонам связи названия. В свойство Name для сторон End1 и End2 введем roleLecturer и roleSubjects соответственно (см. рисунок 8.1).

6. Заполним свойства Alias созданных элементов, чтобы на диаграмме классов отображались русские названия.

Создание машины состояний

В нашем примере дополним класс Дисциплина новыми возможностями. Каждая дисциплина будет иметь определенный статус. Возможно добавление дисциплины преподавателю (состояние Выбран преподаватель).

После чего она может быть принята на рассмотрение (состояние Выбрана дисциплина), затем либо назначена преподавателю (состояние Назначена), либо отклонена (состояние Отклонена). Состояние Отклонена дисциплина возможно также, если сумма часов предложенных преподавателю дисциплин превысит допустимую нагрузку на преподавателя. В случае отклонения дисциплины рассматриваемый экземпляр класса Дисциплина автоматически удаляется. Покажем, как реализовать такой алгоритм исключительно на уровне модели, не прибегая к кодированию на языке Delphi.

1. Перейдем в модельное пространство проекта. Выберем элемент clSubject и с помощью контекстного меню дадим команду Add > ECO State Machine. В рамках Проектировщика откроется новое, первоначально пустое окно. В нем будет построена соответствующая модель. Назовем модель StatesOfSubject.

2. Поместим на диаграмму начальное состояние Initial.

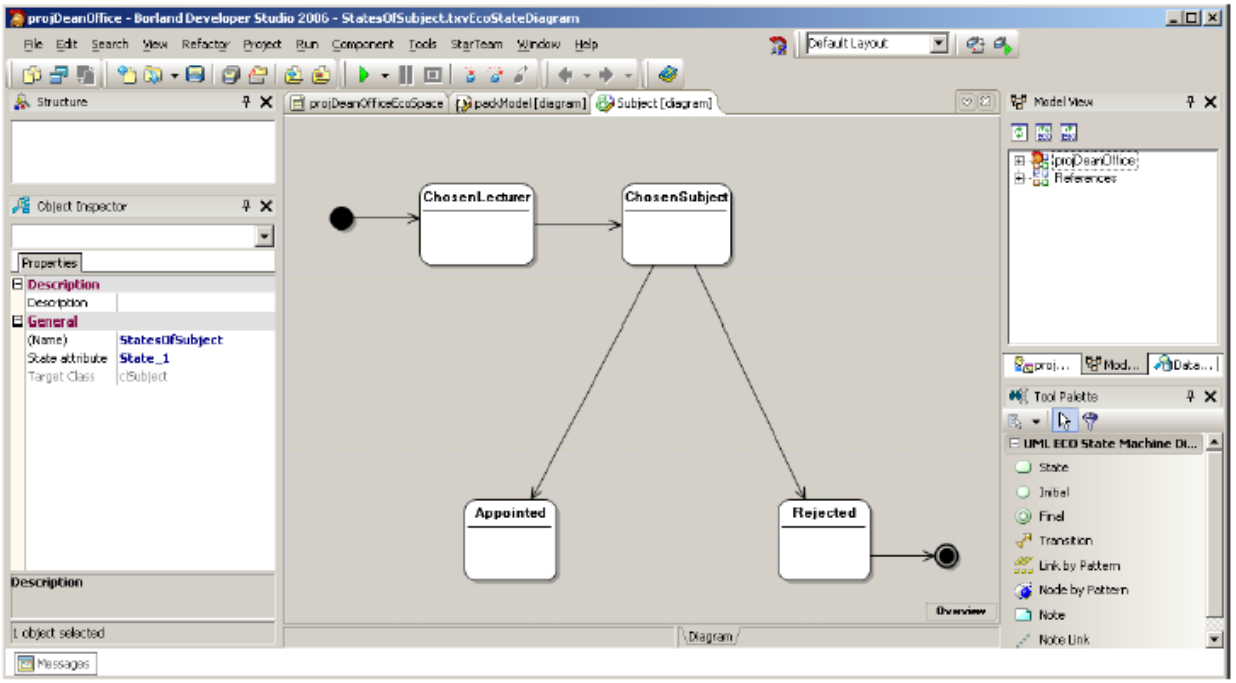


Рисунок 8.2 - Диаграмма машины состояний

3. Добавим на диаграмму четыре основных состояния дисциплины с помощью инструмента State. Дадим им имена: ChosenLecturer (Выбран преподаватель), ChosenSubject (Выбрана дисциплина), Appointed (Назначена), Rejected (Отклонена).

4. Поместим на диаграмму конечное состояние Final.

5. Согласно проектной логике свяжем эти состояния отношениемпереходом с помощью инструмента Transition (рисунок 8.2).

6. Вернемся на диаграмму классов ECO. Видно, что в классе Дисциплина появилось новое свойство State_1 типа String (назовем его SubjectState), обозначающее нашу машину состояний. Теперь можно ввести выражение OCL для вычисляемого поля Количество рабочих часов (LectAmountHours) класса Преподаватель. В свойстве Derivation OCL этого атрибута создадим с помощью встроенного OCL-редактора выражение:

self.roleSubjects->select(SubjectState='Appointed').SubjAmountHours->sum

Оно означает, что в поле Количество рабочих часов каждого объекта Преподаватель будет сумма часов тех дисциплин, которые ему уже назначены (поле SubjectState имеет значение Appointed).

7. Для каждого перехода надо задать триггер.

В терминологии модели ECO *триггер* – это действие, которое производится при выполнении определенного условия и модифицирует текущее состояние (вызывает переход к другому состоянию).

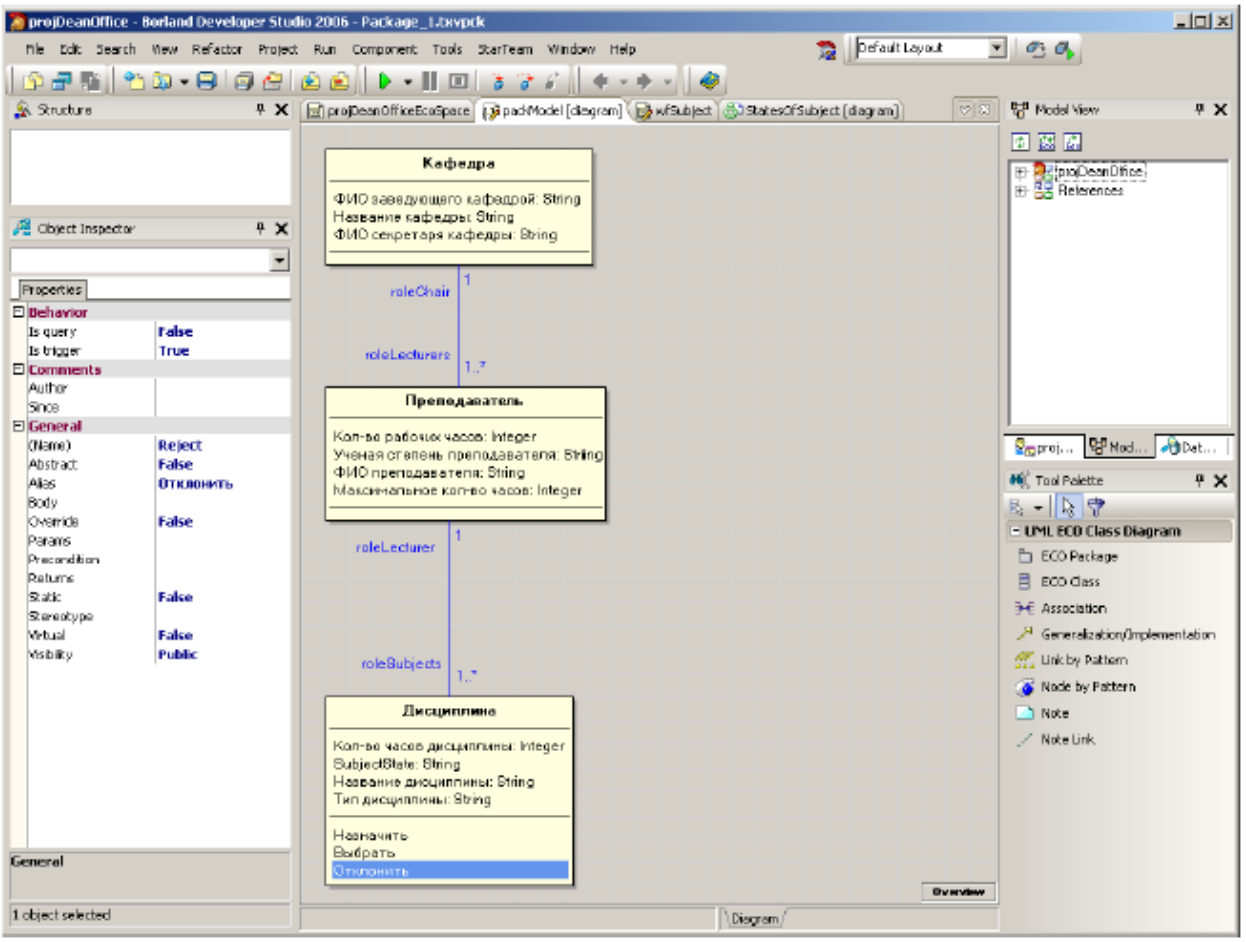


Рисунок 8.3. Список возможных состояний класса Дисциплина

Добавим в класс Дисциплина три триггера. Каждый триггер добавляют командой контекстного меню Add > Trigger. Дадим этим триггерам соответствующие имена: Choose, Appoint и Reject. Их свойства Alias заполним русскоязычными названиями: Выбрать, Назначить и Отклонить (рисунок 8.3).

Для действия Выбран преподаватель создавать триггер не нужно.

Согласно нашей модели считается, что экземпляр класса Дисциплина получает этот статус, как только он сформирован.

8. Распределим триггеры по переходам на диаграмме машины состояний. Будем выбирать по очереди каждую связь (переход между состояниями) и выбирать в ее свойстве Trigger нужное имя триггера из раскрывающегося списка.

9. Зададим правило перехода из состояния Выбрана дисциплина в состояние Назначена. Пока что ограничений или дополнительных требований на этот переход не наложено, и пользователь может бесконтрольно менять состояние дисциплины. Правило перехода задается в свойстве Guard (сторожевое условие) в виде условного выражения OCL:

(self.roleLecturer.LectAmountHours+self.SubjAmountHours) <= self.roleLecturer.MaxAmountHours

Данное выражение означает, что сумма рабочих часов преподавателя, претендента на ведение выбранной дисциплины, и количество часов этой дисциплины должно быть меньше или равно максимальному количеству рабочих часов преподавателя. Теперь возможность назначить дисциплину, если сумма рабочих часов превышает допустимое значение, недоступна, пока преподавателю не увеличат максимальное количество часов или не сократят количество часов для дисциплины.

Обновление базы данных

1. Чтобы сгенерировать исходный код ECO в соответствии с построенной моделью, выберем операцию Regenerate ECO source code из консольного меню окна Model View. Также такая операция производится нажатием кнопки над данным окном. Генерация производится автоматически.

2. Так как мы создали новый класс и добавили новые поля в уже существующий,

необходимо обновить подключенную базу данных. Для этого перейдем на закладку настройки объектного пространства projDeanOfficeEcoSpace. Нажмем кнопку Generate Schema в нижней части окна. В текущей базе данных автоматически сформируются таблицы, содержащие поля для хранения модели проекта. При этом все имеющиеся данные в базе удаляются. Операция Evolve Schema позволяет сохранить существующие данные и дополнить базу созданными элементами в соответствии с построенной моделью.

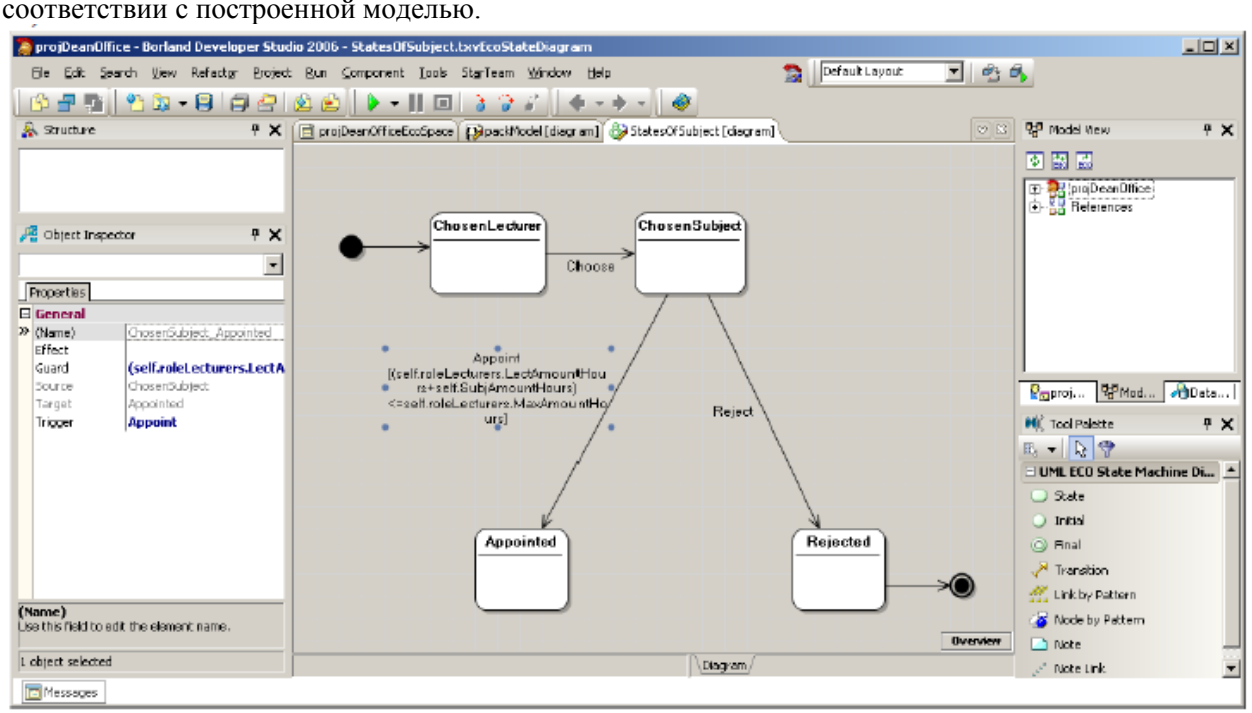


Рисунок 8.4. Добавляем триггеры и сторожевое условие к диаграмме машин состояний

Модификация пользовательского интерфейса

Добавим на форму элементы интерфейса пользователя.

1. Создадим вторую дочернюю форму wfSubject (класс TSubject). Заголовок формы – Дисциплины.

2. Перейдем к вкладке Code. В окне появится код формы. Перед командой Implementation объявим переменную, ответственную за создание дочернего окна Дисциплины:

```
var
callSubj: TSubject;
```

3. Перейдем к Главной форме. Чтобы в Главной форме создать дочернюю, подключим к ней форму wfSubject командой File > Use Unit. В появившемся окне Use Unit выберем форму wfSubject.

4. В главном меню Главной формы добавим подпункт Дисциплины в пункте Подсистемы. На событие выбора пункта меню Дисциплины пропишем следующие операции:

```
callSubj := TSubject.Create(EcoSpace);
callSubj.MdiParent := self;
callSubj.Show;
```

5. Вернемся к форме Дисциплины. Разместим на ней две таблицы: dgLecturer и dgSubject (экземпляры класса DataGrid) для представления экземпляров классов Преподаватель и Дисциплина. Заполним в таблицах свойство CaptionText (название заголовка).

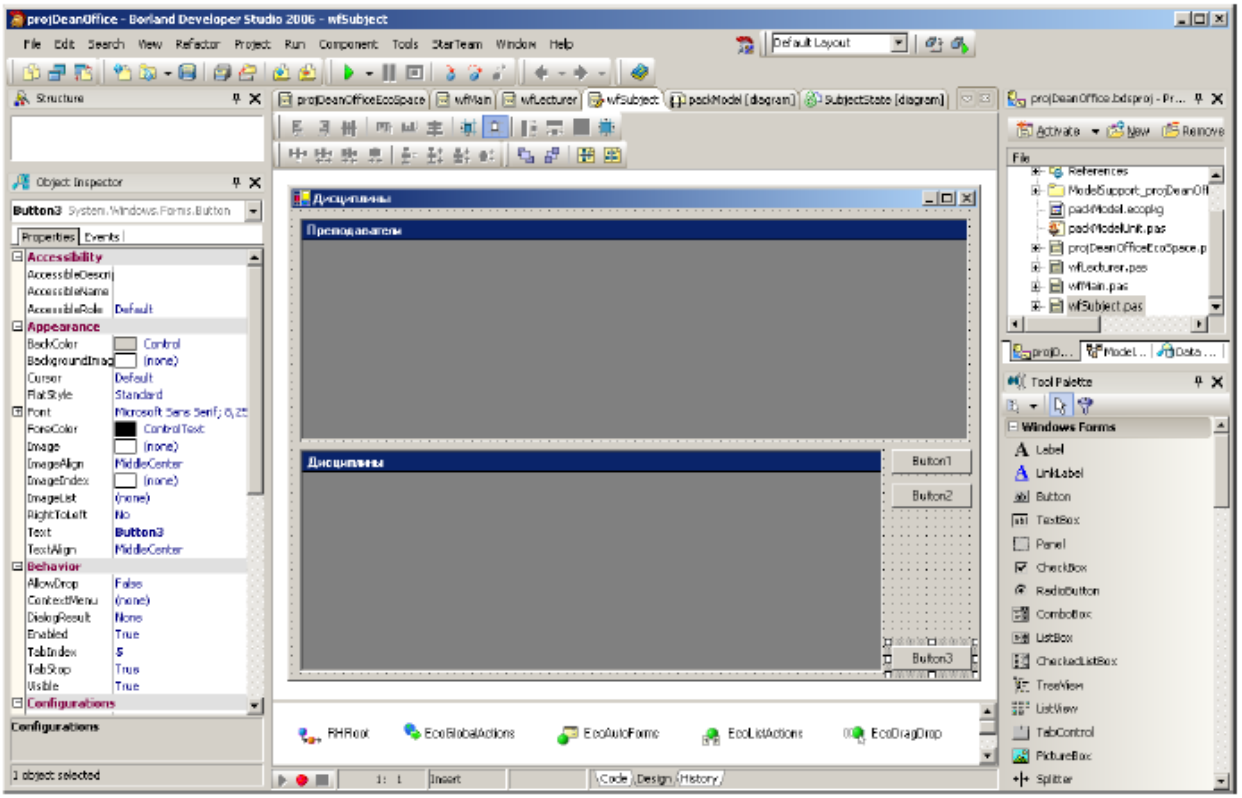


Рисунок 8.5 - Настройка пользовательского интерфейса

6. Добавим две кнопки для работы с таблицей Дисциплины (добавление, удаление объектов) и третью кнопку для обновления базы данных (рисунок 8.5).

Связывание интерфейса с моделью

1. Сформируем три дескриптора ECO: ehLecturer, ehSubject (экземпляры класса ExpressionHandle) для организации доступа к объектам модели и пространства ECO во время работы программы и cmhLecturer (экземпляр класса CurrensyManagerHandle), который будет служить для указания на текущий объект таблицы Преподаватели.

2. Настроим таблицу Дисциплины так, чтобы она отражала список дисциплин, принадлежащих выбранному преподавателю в первой таблице.

Свяжем дескриптор cmhLecturer с таблицей dgLecturer через свойство BindingContext.

Теперь он отслеживает выделенную строку этой таблицы.

3. Настроим дескриптор cmhLecturer на связь с родительским объектом. Зададим ссылку на объект ehLecturer в свойстве RootHandle, определяющем корневой идентификатор ECO.

4. В свойстве RootHandle дескриптора ehSubject выберем значение cmhLecturer. Так задают место данного идентификатора в цепочке доступа к объектному пространству ECO.

5. В качестве выражения OCL введем в свойство Expression объекта ehSubject строку self.roleSubjects. Здесь roleSubjects – это имя роли класса Subject в его ассоциативной связи с классом Преподаватель. Это выражение определяет список дисциплин, предложенных и назначенных преподавателю, который выбран в таблице Преподаватели.

6. Свяжем таблицу Преподаватели с дескриптором ehLecturer через ее свойство DataSource, а таблицу Дисциплины – с дескриптором ehSubject.

Теперь таблицы отображают поля классов Преподаватель и Дисциплина.

Кроме этого, в таблице Дисциплины появилось поле SubjectState, которое будет автоматически отображать текущее состояние дисциплин.

7. Настроим стили таблиц. Воспользуемся свойством таблицы TableStyles. Присвоим полям таблиц русскоязычные названия.

8. Настроим функционирование кнопок. Выделим в окне Проектировщика кнопку Button1. В свойстве EcoListAction выберем значение Add. В свойстве RootHandle выберем подходящий поставщик объектов ECO – идентификатор ehSubject. Свяжем результат действия кнопки (созданный экземпляр класса Дисциплина) с визуальным элементом, отображающим этот экземпляр, в нашем случае – с таблицей dgSubject. Для этого в свойстве BindingContext выберем имя dgSubject. Аналогично настроим кнопку Button2 на операцию удаления. Выберем компонент

EcoListActions, в его свойстве CaptionAdd введем слово Добавить, а в свойстве CaptionDelete – Удалить.

Кнопка автоматически получила название Добавить и Удалить. В свойстве Text кнопки Button3 введем значение Сохранить (см. рисунок 8.6).

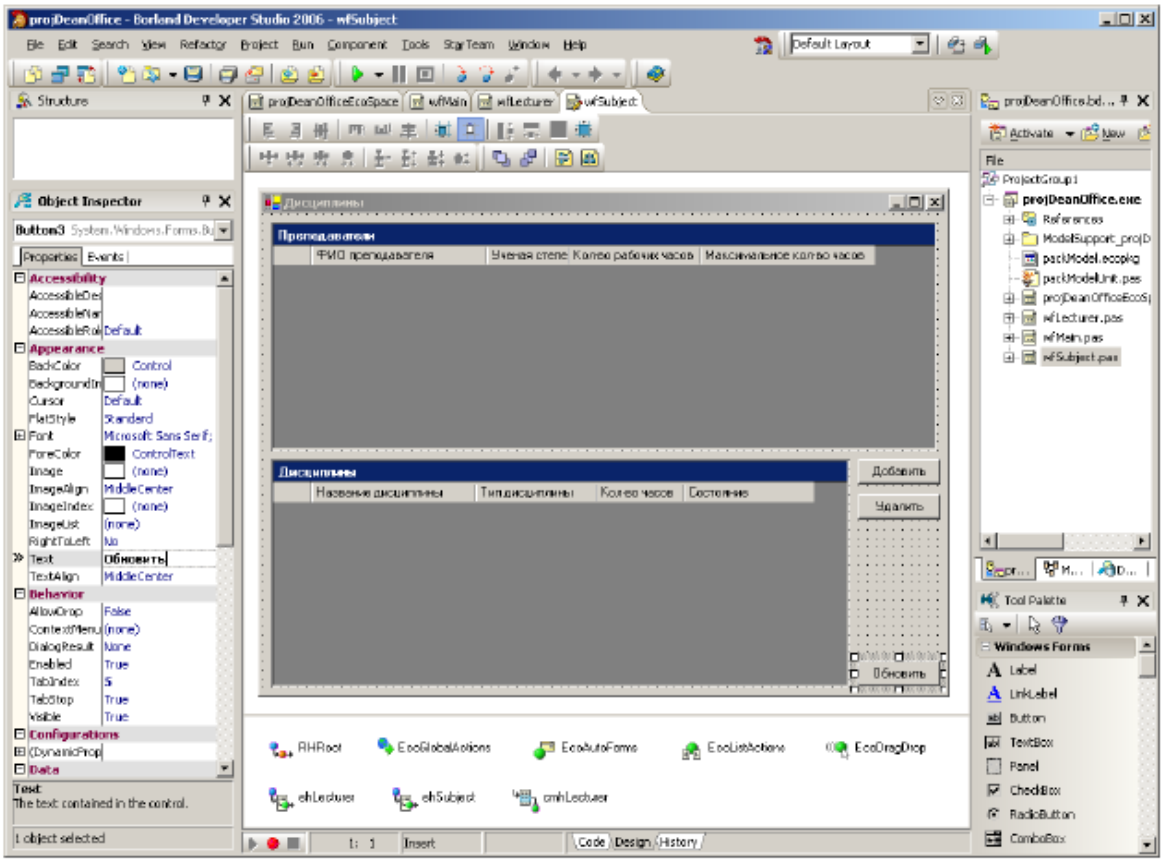


Рисунок 8.6 - Связывание интерфейса с моделью

9. Запустим приложение. Добавим несколько объектов в таблицу Дисциплины для выбранного преподавателя. В поле Состояние таблицы Дисциплины все объекты имеют одинаковое значение ChosenLecturer. На данный момент это состояние неизменяемо. Чтобы иметь возможность изменять состояния, произведем следующую модификацию пользовательского интерфейса.

Применение автоформ

Автоформа ECO предназначена для модификации в отдельном окне объектов ECO, представленных в таблицах. Для вызова автоформы надо задать в свойстве таблицы EcoAutoForm значение True. Если запустить программу и дважды щелкнуть мышью на произвольной строке таблицы, откроется диалоговое окно, содержащее текстовые поля, которые соответствуют столбцам таблицы (атрибутам объекта ECO). В этом окне отображается содержимое полей строки, на которой был выполнен щелчок.

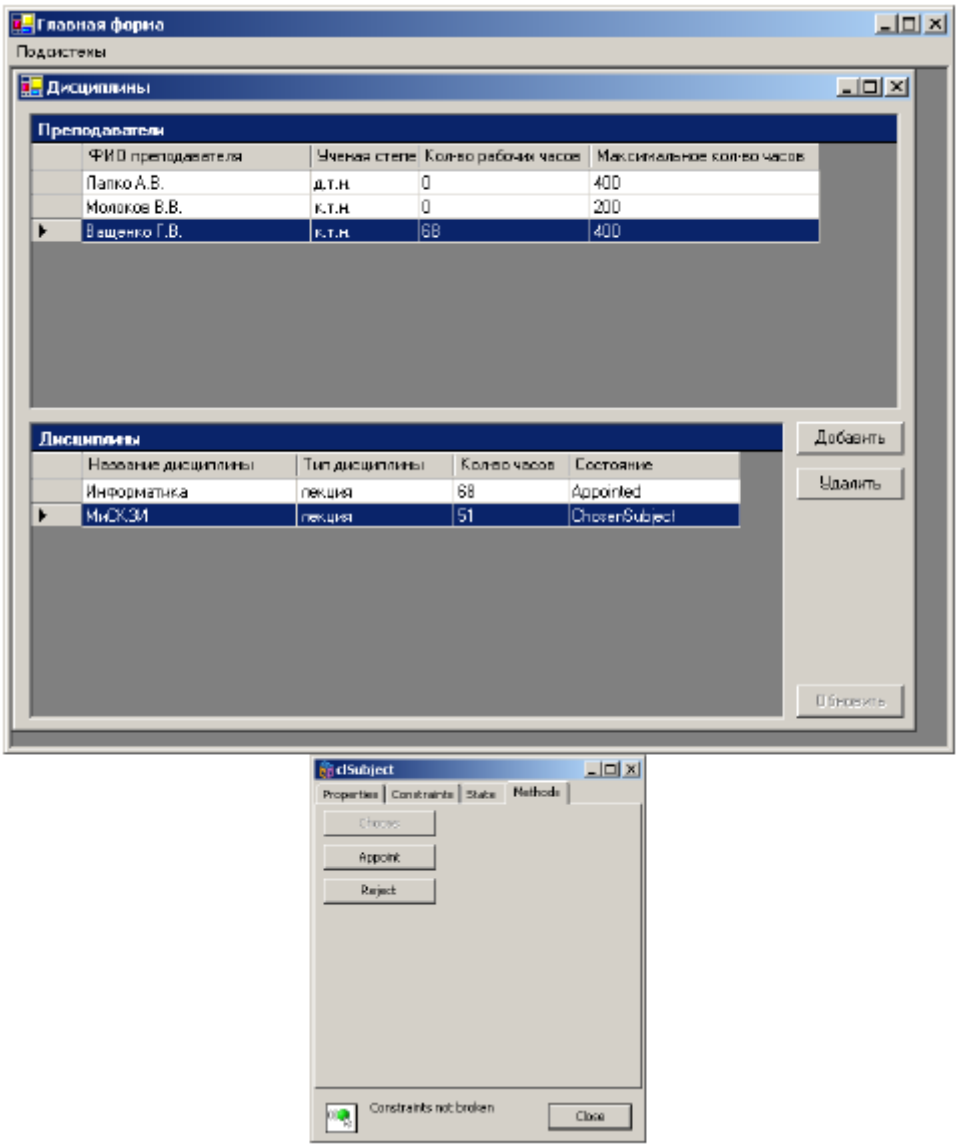


Рисунок 8.7 - Работа с таблицей в режиме автоформы

Для одной таблицы можно открыть несколько окон автоформы одновременно и редактировать значения разных объектов. При этом модифицированные величины корректно отображаются в таблице и фиксируются в объектном пространстве. Данная возможность, как будет показано далее, особо полезна в случаях, когда между объектами таблицы установлены дополнительные связи.

5) Таблицу Дисциплины настроим на работу в режиме автоформы. Для этого в свойстве EcoAutoForm зададим значение True.

6) Запустим приложение. Если щелкнуть в начале произвольной строки таблицы, хранящей список дисциплин, откроется автоформа для ее редактирования. В ней на закладке Methods показаны возможные переходы между состояниями, причем в соответствии с последовательностью переходов: в следующее состояние нельзя перейти, не побывав в предыдущем. Если вавтоформе нажать кнопку Choose (подпись кнопки совпадает с именем соответствующего триггера), список доступных переходов автоматически изменяется. Соответствующие изменения отобразятся и в колонке Состояние таблицы Дисциплины. Причем, переход в состояние Назначить дисциплину ограничен условием: количество рабочих часов преподавателя, которого собираются назначить на ведение выбранной дисциплины, в сумме с количеством часов назначаемой дисциплины должно быть меньше или равно максимальному количеству рабочих часов для преподавателя (см. рисунок 8.7).

Расширение пользовательского интерфейса

Команды-триггеры для удобства можно привязать к пользовательским элементам управления. Создадим в текущем проекте контекстное меню с помощью компонента ContextMenu из категории Components. Пусть в нем будет три пункта: Выбрать, Назначить и Отклонить. Их

необходимо связать с соответствующими триггерами.

1. Добавим на форму компонент ContextMenu, дадим ему имя cmSubject.

2. В свойстве меню RootHandle задается корневой идентификатор ehSubject.

3. Объект управления (таблица Дисциплины), для которого вызывается данное меню, выбирается в свойстве BindingContext.

4. В свойстве EcoListAction выбирается тип действия ECO, выполняемого при выборе данного пункта. Введем значение ExecuteAction (исполняемое выражение OCL).

5. Само выражение OCL следует ввести в свойство ActionExpression. Это выражение формируется с помощью раздела Triggers в редакторе выражений OCL. Так, для пункта Выбрать дисциплину это выражение записывается как строка self.Choose.

6. В свойстве EnabledOCL с помощью этого же редактора формируется выражение OCL – триггерный запрос, который определяет, доступен ли пользователю соответствующий элемент управления (в нашем случае – пункт меню). Если триггер недоступен, то и пункт меню автоматически блокируется. Запрос выбирается в разделе редактора Trigger queries. Например, для пункта меню Назначить он записывается строкой self.Appoint (рисунок 8.8).

7. Выделим таблицу dgSubject. В ее свойстве ContextMenu выберем ссылку на настроенный компонент cmSubject. Привязка и настройка контекстного меню для объектов таблицы Дисциплины закончена.

8. Запустим приложение. Видно, что контекстное меню каждой строки (доступные в ней пункты) автоматически меняется в зависимости от состояния текущей дисциплины (см. рисунок 8.9).

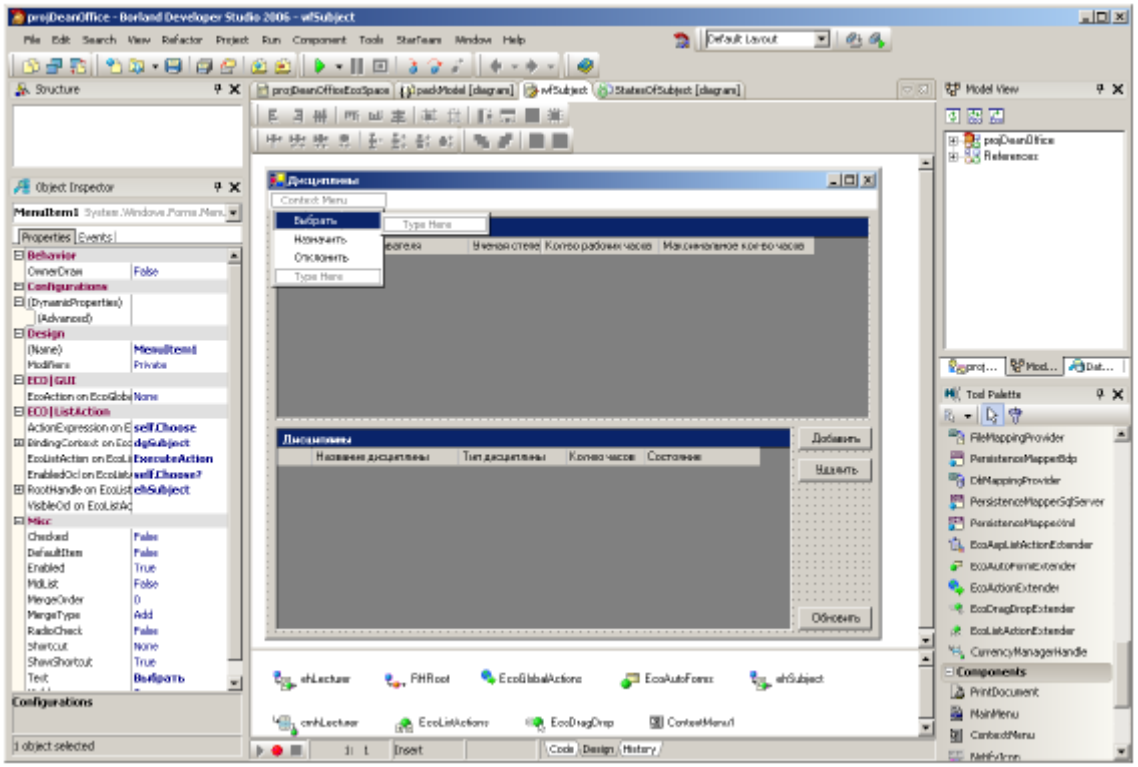


Рисунок 8.8 - Связывание команд-триггеров с компонентом ContextMenu

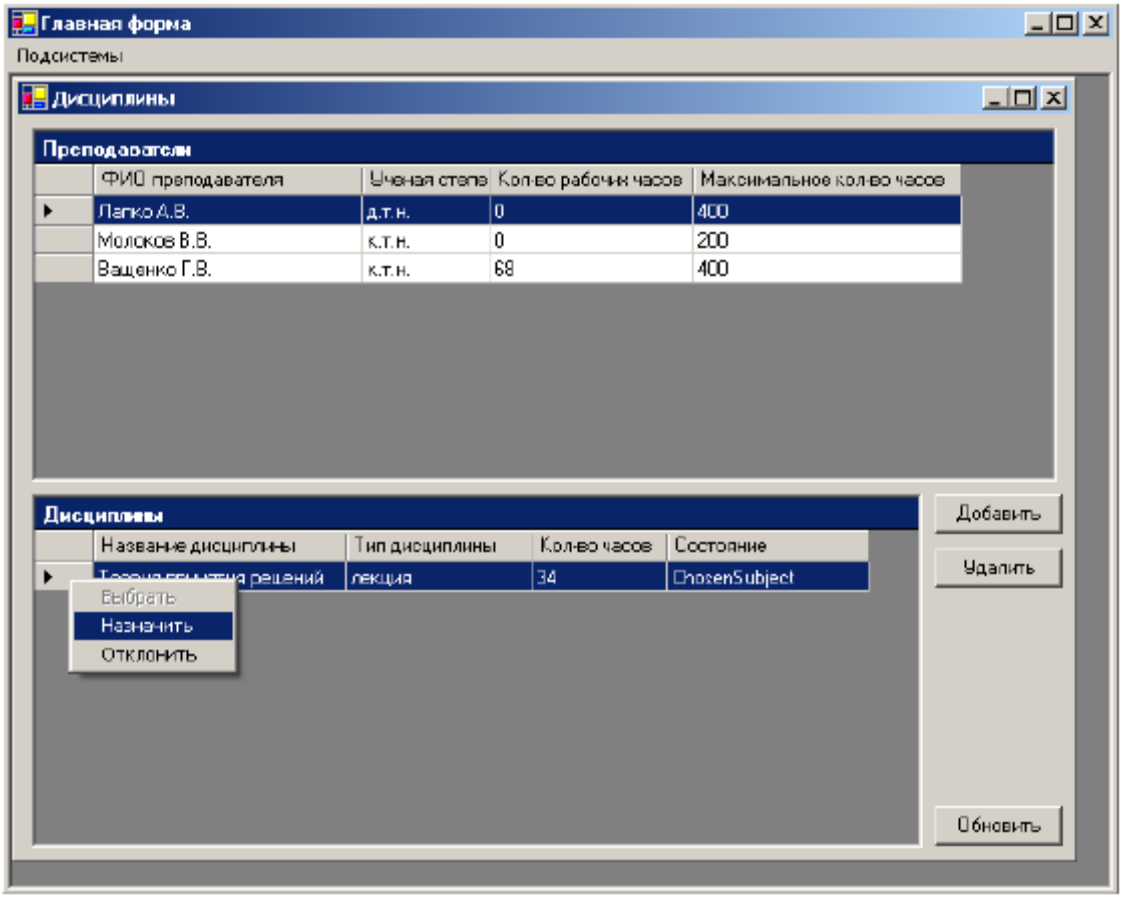


Рисунок 8.9 - Применение контекстного меню в приложении

Задания и порядок выполнения работы

1. Продумать, какой объект из предметной области можно представить в виде последовательности определенных состояний.
2. Расширить модель UML: создать машину состояний для выбранного объекта (или объектов) и модифицировать диаграмму классов.
3. Обновить базу данных, модифицировать пользовательский интерфейс и связать его с моделью.
4. Применить автоформы или контекстное меню.

Контрольные вопросы

1. Что такое автомат?
2. Назовите базовые правила работы автоматов в языке UML.
3. Для чего предназначены диаграммы состояний?
4. Как строятся диаграммы машин состояний?
5. Как настраиваются триггеры в диаграммах состояний?
6. Как задаются правила перехода на языке OCL для триггеров?
7. Как связать триггеры с элементами пользовательского интерфейса?

Работа с литературой:

Рекомендуемые источники информации (№ источника)			
Основная	Дополнительная	Методическая	Интернет-ресурсы
1-3	1-3	1-3	1-2

Оценочные средства: контрольные вопросы

5. КРИТЕРИИ ОЦЕНИВАНИЯ КОМПЕТЕНЦИЙ

Оценка «отлично» выставляется студенту, если теоретическое содержание курса освоено полностью, без пробелов; исчерпывающе, последовательно, четко и логически стройно излагает материал; свободно справляется с задачами, вопросами и другими видами применения знаний; использует в ответе дополнительный материал все предусмотренные программой задания выполнены, качество их выполнения оценено числом баллов, близким к максимальному; анализирует полученные результаты; проявляет самостоятельность при выполнении заданий.

Оценка «хорошо» выставляется студенту, если теоретическое содержание курса освоено полностью, необходимые практические компетенции в основном сформированы, все предусмотренные программой обучения учебные задания выполнены, качество их выполнения достаточно высокое. Студент твердо знает материал, грамотно и по существу излагает его, не допуская существенных неточностей в ответе на вопрос.

Оценка «удовлетворительно» выставляется студенту, если теоретическое содержание курса освоено частично, но пробелы не носят существенного характера, большинство предусмотренных программой заданий выполнено, но в них имеются ошибки, при ответе на поставленный вопрос студент допускает неточности, недостаточно правильные формулировки, наблюдаются нарушения логической последовательности в изложении программного материала.

Оценка «неудовлетворительно» выставляется студенту, если он не знает значительной части программного материала, допускает существенные ошибки, неуверенно, с большими затруднениями выполняет практические работы, необходимые практические компетенции не сформированы, большинство предусмотренных программой обучения учебных заданий не выполнено, качество их выполнения оценено числом баллов, близким к минимальному.

6.МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ, ОПРЕДЕЛЯЮЩИЕ ПРОЦЕДУРЫ ОЦЕНИВАНИЯ ЗНАНИЙ, УМЕНИЙ, НАВЫКОВ И (ИЛИ) ОПЫТА ДЕЯТЕЛЬНОСТИ, ХАРАКТЕРИЗУЮЩИХ ЭТАПЫ ФОРМИРОВАНИЯ КОМПЕТЕНЦИЙ

Промежуточная аттестация проводится в форме экзамена.

Процедура проведения экзамена осуществляется в соответствии с Положением о проведении текущего контроля успеваемости и промежуточной аттестации обучающихся по образовательным программам высшего образования в СКФУ.

В экзаменационный билет включаются три вопроса: по одному вопросу из категорий «знать, уметь, владеть».

Для подготовки по билету отводится 30 минут.

При подготовке к ответу студенту предоставляется право пользования учебно-методическим комплексом дисциплины.

Текущая аттестация студентов проводится преподавателями, ведущими лабораторные занятия по дисциплине.

Практическое выполнение задания со студентом должно представлять собой связанный, логически последовательныйобмен сообщениями на заданную тему, показывать его умение применять определения, правила в конкретных случаях. Основные требования к собеседованию: полноту и правильность ответа;степень осознанности, понимания изученного;языковое оформление ответа.

Метод case-study (от английского case – случай, ситуация) – метод активного проблемно-ситуационного анализа, основанный на обучении путем решения конкретных задач – ситуаций (решение кейсов). Относится к неигровым имитационным активным методам обучения. Непосредственная цель метода case-study – совместными усилиями группы студентов проанализировать ситуацию – case, возникающую при конкретном положении дел, и выработать практическое решение; окончание процесса – оценка предложенных алгоритмов и выбор лучшего в контексте поставленной проблемы. Технология метода заключается в следующем: по определенным правилам разрабатывается модель конкретной ситуации, произошедшей в реальной жизни, и отражается тот комплекс знаний и практических навыков, которые студентам нужно получить; при этом преподаватель выступает в роли ведущего, генерирующего вопросы, фиксирующего ответы, поддерживающего дискуссию, т.е. в роли диспетчера процесса сотворчества. Проверка и оценка знаний должны проводиться согласно дидактическим принципам

обучения. При этом выделяются следующие требования к оцениванию: объективность – создание условий, в которых бы максимально точно выявлялись знания обучаемых, предъявление к ним единых требований, справедливое отношение к каждому; обоснованность оценок – их аргументация; систематичность – важнейший психологический фактор, организующий и дисциплинирующий студентов, формирующий настойчивость и устремленность в достижении цели; всесторонность и оптимальность.

7. УЧЕБНО-МЕТОДИЧЕСКОЕ И ИНФОРМАЦИОННОЕ ОБЕСПЕЧЕНИЕ ДИСЦИПЛИНЫ

7.1.1. Перечень основной литературы:

1. Битюцкая, Н. И. Разработка программных приложений : лабораторный практикум / Н. И. Битюцкая. — Ставрополь : Северо-Кавказский федеральный университет, 2015. — 140 с.

2. Абрамов, Г. В. Проектирование и разработка информационных систем : учебное пособие для СПО / Г. В. Абрамов, И. Е. Медведкова, Л. А. Коробова. — Саратов : Профобразование, 2020. — 169 с. — ISBN 978-5-4488-0730-5. — Текст : электронный // Цифровой образовательный ресурс IPR SMART : [сайт]. — URL: https://www.iprbookshop.ru/88888.html (дата обращения: 10.04.2022). — Режим доступа: для авторизир. пользователей. - DOI: https://doi.org/10.23682/88888

7.1.2. Перечень дополнительной литературы:

1. Баженова, И. Ю. Основы проектирования приложений баз данных : учебное пособие / И. Ю. Баженова. — 3-е изд. — Москва : Интернет-Университет Информационных Технологий (ИНТУИТ), Ай Пи Ар Медиа, 2020. — 324 с. — ISBN 978-5-4497-0682-9. — Текст : электронный // Цифровой образовательный ресурс IPR SMART : [сайт]. — URL: https://www.iprbookshop.ru/97569.html (дата обращения: 10.04.2022). — Режим доступа: для авторизир. пользователей

2. Гвозденко, Н. П. Разработка блок-схем алгоритмов : учебное пособие / Н. П. Гвозденко, С. А. Суслова. — Липецк : Липецкий государственный технический университет, ЭБС АСВ, 2021. — 59 с. — ISBN 978-5-00175-055-0. — Текст : электронный // Цифровой образовательный ресурс IPR SMART : [сайт]. — URL: https://www.iprbookshop.ru/116169.html (дата обращения: 10.04.2022). — Режим доступа: для авторизир. Пользователей

3. Савельев, А. О. Проектирование и разработка веб-приложений на основе технологий Microsoft : учебное пособие / А. О. Савельев, А. А. Алексеев. — 4-е изд. — Москва : Интернет-Университет Информационных Технологий (ИНТУИТ), Ай Пи Ар Медиа, 2022. — 418 с. — ISBN 978-5-4497-1650-7. — Текст : электронный // Цифровой образовательный ресурс IPR SMART : [сайт]. — URL: https://www.iprbookshop.ru/120486.html (дата обращения: 10.04.2022). — Режим доступа: для авторизир. пользователей

7.2.Перечень учебно-методического обеспечения самостоятельной работы обучающихся по дисциплине:

3. Методические указания по выполнению контрольной работы по дисциплине «Технология разработки программного обеспечения».

4. Методические рекомендации для студентов по организации самостоятельной работы по дисциплине «Технология разработки программного обеспечения».

7.3. Перечень ресурсов информационно-телекоммуникационной сети «Интернет», необходимых для освоения дисциплины:

1. http://el.ncfu.ru/ – система управления обучением ФГАОУ ВО СКФУ. Дистанционная поддержка дисциплины «Цифровая грамотность и обработка данных»
2. http://www.un.org - Сайт ООН Информационно-коммуникационные технологии
3. http://www.intuit.ru – Интернет-Университет Компьютерных технологий

8. Перечень информационных технологий, используемых при осуществлении образовательного процесса по дисциплине (модулю), включая перечень программного обеспечения и информационных справочных систем

При чтении лекций используется компьютерная техника, демонстрации презентационных мультимедийных материалов. На семинарских и практических занятиях студенты представляют презентации, подготовленные ими в часы самостоятельной работы.

Информационные справочные системы:

Информационно-справочные и информационно-правовые системы, используемые при изучении дисциплины:

1	КонсультантПлюс - http://www.consultant.ru/

Программное обеспечение:

1	Операционная система: Microsoft Windows 8: 2013-02(3000). Бессрочная лицензия. Договор № 01-эа/13 от 25.02.2013. Окончание бесплатной поддержки – 2024-01 ИЛИ Операционная система: Microsoft Windows 10: 2016-08(20), 2017-10(67), 2018-01(18), 2018-04(6), 2018-05(6), 2019-02(7). Бессрочная лицензия. Договоры № 27-эа/16 от 02.08.2016. и № 0321100021117000009_229123 от 10.10.2017. На текущий момент окончания поддержки не анонсировано.
2	Базовый пакет программ Microsoft Office (Word, Excel, PowerPoint). MicrosoftOfficeStandard 2013: договор № 01-эа/13 от 25.02.2013г., Лицензирование Microsoft Office https://support.microsoft.com/ru-ru/lifecycle/search/16674 Дата начала жизненного цикла 09.01.2013г.; набор обновлений Office 2013 Service Pack1 Дата начала жизненного цикла 25.02.2014г., Дата окончания основной фазы поддержки 10.04.2018; Дополнительная дата окончания поддержки 11.04.2024г.

9. Описание материально-технической базы, необходимой для осуществления образовательного процесса по дисциплине (модулю)

Лабораторные занятия	Учебная аудитория для проведения занятий семинарского типа (практических работ) — компьютерная аудитория	Количество рабочих мест – 12 Оборудование: Персональные компьютеры, объединенные в локальную сеть и имеющие выход в интернет
Самостоятельная работа	Помещения для самостоятельной работы	Компьютеры с выходом в Интернет и обеспечением доступа в электронную информационно-образовательную среду ИСУ СКФУ.

Учебные аудитории для проведения учебных занятий, оснащены оборудованием и техническими средствами обучения. Помещения для самостоятельной работы обучающихся, оснащенные компьютерной техникой с возможностью подключения к сети "Интернет" и обеспечением доступа к электронной информационно-образовательной среде. Специализированная мебель и технические средства обучения, служащие для представления учебной информации.

Материально-техническая база обеспечивает проведение всех видов дисциплинарной и междисциплинарной подготовки, лабораторной, научно-исследовательской работы обучающихся (переносной ноутбук, переносной проектор, компьютеры с необходимым программным обеспечением и выходом в интернет).

Помещения для самостоятельной работы обучающихся оснащены компьютерной техникой с возможностью подключения к сети «Интернет» и обеспечением доступа в электронную информационно-образовательную среду образовательной организации.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Пятигорский институт (филиал) СКФУ

Методические указания
по организации самостоятельной работы обучающихся
по дисциплине «**ТЕХНОЛОГИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ»**

Направление подготовки/специальность **09.04.02 Информационные системы и технологии**
Направленность (профиль)/специализация **Технологии работы с данными и знаниями, анализ информации**
Форма обучения **очная, заочная**
Год начала обучения **2024**
Изучается в 3 семестре

СОДЕРЖАНИЕ

1. ЦЕЛЬ И ЗАДАЧИ ОСВОЕНИЯ ДИСЦИПЛИНЫ

Целью изучения дисциплины «Технологии разработки программного обеспечения» учебного плана подготовки магистров по направлению подготовки 09.04.02 «Информационные системы и технологии», является получение компетенций, достаточных для анализа требований к программным системам, их документирования, проектирования, разработки, тестирования, внедрения, управления программными проектами и управления качеством разработки программных систем.

Задачами курса является приобретение и развитие знаний, умений и навыков для производственно-технологической, организационно-управленческой, проектной и научно-исследовательской деятельности.

2. КОМПЕТЕНЦИИ ОБУЧАЮЩЕГОСЯ, ФОРМИРУЕМЫЕ В РЕЗУЛЬТАТЕ ИЗУЧЕНИЯ ДИСЦИПЛИНЫ

Код, формулировка компетенции	Код, формулировка индикатора	Планируемые результаты обучения по дисциплине (модулю), характеризующие этапы формирования компетенций, индикаторов
ПК-2 способен создания технической документации информационно-методического и маркетингового назначения в сфере информационных технологий и систем	**ИД-1 ПК-2 С**оздает техническую документации информационно-методического назначения в сфере информационных технологий и систем **ИД-2 ПК-2** Применяет техническую документацию для создания информационных технологий и систем **ИД-3 ПК-2** Использует техническую документацию для решения маркетинговых задач в сфере информационных технологий и систем;	Проводит предпроектное обследование объекта проектирования, анализ научно-технической информации, отечественного и зарубежного опыта
ПК-4 способен выполнять разработку систем управления базами данных, операционных систем, организацию разработки системного программного обеспечения, интеграция разработанного системного программного обеспечения	**ИД-1 ПК-4** Выполняет разработку систем управления базами данных. **ИД-2 ПК-4** Проводить непосредственное руководство процессами разработки программного обеспечения; **ИД-3 ПК-4** Проводить организацию разработки системного программного обеспечения, интеграцию разработанного системного программного обеспечения.	Проводит анализ разработанных программных приложений
ПК-6 способен проводить организационное сопровождение разработки, отладки, модификации и поддержки информационных технологий и систем	**ИД-1 ПК-6** Проводить организационное сопровождение процессом разработки ПО **ИД-2 ПК-6** Выполняет управление проектами в области ИТ любого масштаба в условиях высокой неопределенности **ИД-3 ПК-6** Проводить отладку, модификацию и поддержку информационных технологий и систем	Проводит техническое проектирование и сопровождение программных приложений
ПК-10 способен выполнять управление аналитическими работами и подразделением	**ИД-1 ПК-10** Выполняет разработку новых инструментов и методов управления проектами в области ИТ. **ИД-2 ПК-10** Проводить разработку новых инструментов; **ИД-3 ПК-10** Использует методы управления проектами в области	Разрабатывает отдельные компоненты информационной системы

	ИТ	

3.ТЕХНОЛОГИЧЕСКАЯ КАРТА САМОСТОЯТЕЛЬНОЙ РАБОТЫ ОБУЧАЮЩЕГОСЯ ОЧНОЙ ФОРМЫ ОБУЧЕНИЯ

Коды реализуемых компетенций	Вид деятельности студентов	Средства и технологии оценки	Объем часов, в том числе (астр.)		
			СРС	Контактная работа с преподавателем	Всего
3 семестр					
ИД-1ПК-2, ИД-2 ПК-2, ИД-1 ПК-4, ИД-2 ПК-4, ИД-1 ПК-9, ИД-2 ПК-9, ИД-1 ПК-10, ИД-2 ПК-10, ИД-1 ПК-12, ИД-2 ПК-12	Самостоятельное изучение литературы и источников	Собеседование	32,94	3,66	36,6
ИД-1ПК-2, ИД-2 ПК-2, ИД-1 ПК-4, ИД-2 ПК-4, ИД-1 ПК-9, ИД-2 ПК-9, ИД-1 ПК-10, ИД-2 ПК-10, ИД-1 ПК-12, ИД-2 ПК-12	Подготовка к лабораторным занятиям	Собеседование	4,86	0,54	5,4
ИД-1ПК-2, ИД-2 ПК-2, ИД-1 ПК-4, ИД-2 ПК-4, ИД-1 ПК-9, ИД-2 ПК-9, ИД-1 ПК-10, ИД-2 ПК-10, ИД-1 ПК-12, ИД-2 ПК-12	Вопросы к экзамену	Собеседование	9	1	10
ИД-1ПК-2, ИД-2 ПК-2, ИД-1 ПК-4, ИД-2 ПК-4, ИД-1 ПК-9, ИД-2 ПК-9, ИД-1 ПК-10, ИД-2 ПК-10, ИД-1 ПК-12, ИД-2 ПК-12	Оценочные средства для курсовой работы	Собеседование	18	2	20
		Итого за 3 семестр	**64,8**	**7,2**	**72**
		Итого	**64,8**	**7,2**	**72**

ЗАОЧНОЙ ФОРМЫ ОБУЧЕНИЯ

Коды реализуемых компетенций	Вид деятельности студентов	Средства и технологии оценки	Объем часов, в том числе (астр.)		
			СРС	Контактная работа с преподавателем	Всего
4 семестр					
ИД-1ПК-2, ИД-2 ПК-2, ИД-1 ПК-4, ИД-2 ПК-4, ИД-1 ПК-9, ИД-2 ПК-9, ИД-1 ПК-10, ИД-2 ПК-10, ИД-1 ПК-12, ИД-2 ПК-12	Самостоятельное изучение литературы и источников	Собеседование	82,08	9,12	91,2
ИД-1ПК-2, ИД-2 ПК-2, ИД-1 ПК-4, ИД-2 ПК-4, ИД-1 ПК-9, ИД-2 ПК-9, ИД-1 ПК-10, ИД-2 ПК-10, ИД-1 ПК-12, ИД-2 ПК-12	Подготовка к лабораторным занятиям	Собеседование	1,62	0,18	1,8
ИД-1ПК-2, ИД-2 ПК-2, ИД-1 ПК-4, ИД-2 ПК-4, ИД-1 ПК-9, ИД-2 ПК-9, ИД-1 ПК-10, ИД-2 ПК-10, ИД-1 ПК-12, ИД-2 ПК-12	Вопросы к экзамену	Собеседование	9	1	10
ИД-1ПК-2, ИД-2 ПК-2, ИД-1 ПК-4, ИД-2 ПК-4, ИД-1 ПК-9, ИД-2 ПК-9, ИД-1 ПК-10, ИД-2 ПК-10, ИД-1 ПК-12, ИД-2 ПК-12	Оценочные средства для курсовой работы	Собеседование	18	2	20
		Итого за 4 семестр	**110,7**	**12,3**	**123**
		Итого	**110,7**	**12,3**	**123**

4. СОДЕРЖАНИЕ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

Тема самостоятельного изучения № 1
Введение в технологии разработки программного обеспечения

Вид деятельности студентов: самостоятельное изучение литературы
Итоговый продукт самостоятельной работы: конспект
Средства и технологии оценки: собеседование
Краткие теоретические сведения

1 ТЕХНОЛОГИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ
1.1 Основные этапы развития технологии разработки

Технологией программирования называют совокупность методов и средств, используемых в процессе разработки программного обеспечения. Как любая другая технология, технология программирования представляет собой набор технологических инструкций, включающих:

- указание последовательности выполнения технологических операций;
- перечисление условий, при которых выполняется та или иная операция;
- описания самих операций, где для каждой операции определены исходные данные, результаты, а также инструкции, нормативы, стандарты, критерии, методы оценки и т. п. (см. рисунок 1.1).

Рисунок 1.1 - Структура описания технологической операции

Кроме набора операций и их последовательности, технология также определяет способ описания проектируемой системы, точнее модели, используемой на конкретном этапе разработки.

Различают технологии, используемые на конкретных этапах разработки или для решения отдельных задач этих этапов, и технологии, охватывающие несколько этапов или весь процесс разработки. В основе первых, как правило, лежит ограниченно применимый *метод,* позволяющий решить конкретную задачу. В основе вторых – базовый метод или *подход,* определяющий совокупность методов, используемых на разных этапах разработки, или проектируемой системы, точнее модели, используемой на конкретном этапе разработки.

Чтобы разобраться в существующих технологиях программирования и определить основные тенденции их развития, целесообразно рассматривать эти технологии в историческом контексте, выделяя основные этапы развития программирования как науки.

1.1.1 Этап 1. «Стихийное» программирование

Этот этап охватывает период от момента появления первых вычислительных машин до середины 60-х гг. XX в. В это время практически отсутствовали технологии разработки программного обеспечения и программирование фактически было искусством. Первые

программы имели простейшую структуру. Они состояли из собственно программы на машинном языке и обрабатываемых ею данных (рисунок 1.2). Сложность программ в машинных кодах ограничивалась способностью программиста одновременно мысленно отслеживать последовательность выполняемых операций и местонахождение данных при программировании.

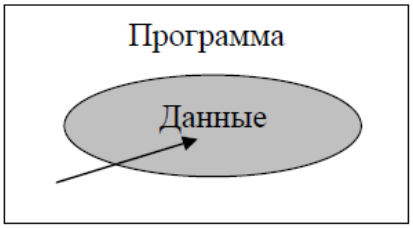


Рисунок 1.2. Структура первых программ

Появление ассемблеров позволило вместо двоичных или шестнадцатеричных кодов использовать символические имена данных и мнемоники ко- дов операций. В результате программы стали более «читаемыми».

Создание языков программирования высокого уровня, таких как FORTRAN и ALGOL, существенно упростило программирование вычислений, снизив уровень детализации операций. Это, в свою очередь, позволило увеличить сложность программ.

Революционным было появление в языках средств, позволяющих оперировать подпрограммами. (Идея написания подпрограмм появилась гораздо раньше, но отсутствие средств поддержки в первых языковых средствах существенно снижало эффективность их применения.) Подпрограммы можно было сохранять и использовать в других программах. В результате были созданы огромные библиотеки расчетных и служебных подпрограмм, которые по мере надобности вызывались из разрабатываемой программы.

Типичная программа того времени состояла из основной программы, области глобальных данных и набора подпрограмм (в основном библиотечных), выполняющих обработку всех данных или их части (рисунок 1.3).

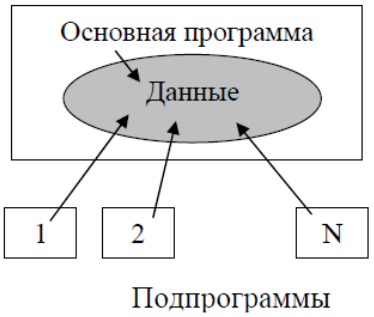


Рисунок 1.3 - Принцип работы программ с глобальной областью данных

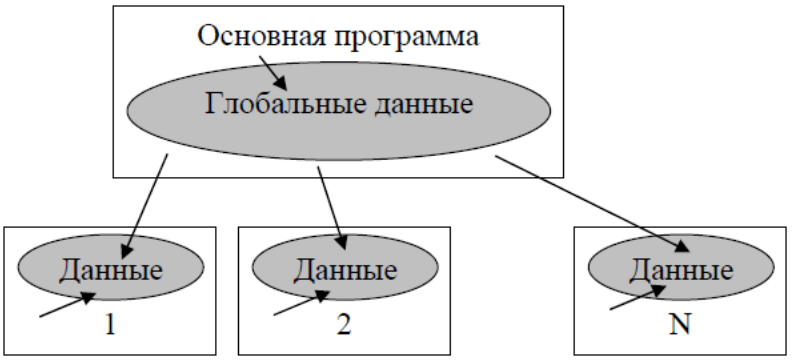


Рисунок 1.4 - Принцип работы программы,
использующей подпрограммы с локальными данными

Слабым местом такой архитектуры было то, что при увеличении количества подпрограмм возрастала вероятность искажения части глобальных данных какой-либо подпрограммой. Например, подпрограмма поиска корней уравнения на заданном интервале по методу деления

отрезка пополам меняет величину интервала. Если при выходе из подпрограммы не предусмотреть восстановления первоначального интервала, то в глобальной области окажется неверное значение интервала. Чтобы сократить количество таких ошибок, было предложено в подпрограммах размещать локальные данные (рисунок 1.4).

Сложность разрабатываемого программного обеспечения при использовании подпрограмм с локальными данными по-прежнему ограничивалась возможностью программиста отслеживать процессы обработки данных, но уже на новом уровне. Однако появление средств поддержки подпрограмм позволило осуществлять разработку программного обеспечения нескольким программистам параллельно.

В начале 60-х гг. XX в. разразился «кризис программирования». Он выражался в том, что фирмы, взявшиеся за разработку сложного программного обеспечения такого, как операционные системы, срывали все сроки завершения проектов. Проект устаревал раньше, чем был готов к внедрению, увеличивалась его стоимость, и в результате многие проекты так никогда и не были завершены.

Объективно все это было вызвано несовершенством технологии программирования. Прежде всего, стихийно использовалась разработка «снизу- вверх» – подход, при котором вначале проектировали и реализовывали сравнительно простые подпрограммы, из которых затем пытались построить сложную программу. В отсутствии четких моделей описания подпрограмм и методов их проектирования создание каждой подпрограммы превращалось в непростую задачу, интерфейсы подпрограмм получались сложными, и при сборке программного продукта выявлялось большое количество ошибок согласования. Исправление таких ошибок, как правило, требовало серьезного изменения уже разработанных подпрограмм, что еще более осложняло ситуацию, так как при этом в программу часто вносились новые ошибки, которые также необходимо было исправлять... В конечном итоге процесс тестирования и отладки программ занимал более 80 % времени разработки, если вообще когда-нибудь заканчивался. На повестке дня самым серьезным образом стоял вопрос разработки технологии создания сложных программных продуктов, снижающей вероятность ошибок проектирования.

Анализ причин возникновения большинства ошибок позволил сформулировать новый подход к программированию, который был назван «структурным».

1.1.2 Этап 2. Структурный подход к программированию (60–70-е гг. XX в.)

Структурный подход к программированию представляет собой совокупность рекомендуемых технологических приемов, охватывающих выполнение всех этапов разработки программного обеспечения. В основе структурного подхода лежит декомпозиция (разбиение на части) сложных систем с целью последующей реализации в виде отдельных небольших подпрограмм. С появлением других принципов декомпозиции (объектного, логического и т. д.) данный способ получил название «процедурной декомпозиции».

В отличие от используемого ранее процедурного подхода к декомпозиции структурный подход требовал представления задачи в виде иерархии подзадач простейшей структуры. Проектирование, таким образом, осуществлялось «сверху-вниз» и подразумевало реализацию общей идеи, обеспечивая проработку интерфейсов подпрограмм. Одновременно вводились ограничения на конструкции алгоритмов, рекомендовались формальные модели их описания, а также специальный метод проектирования алгоритмов – метод пошаговой детализации.

Поддержка принципов структурного программирования была заложена в основу так называемых процедурных языков программирования. Как правило, они включали основные «структурные» операторы передачи управления, поддерживали вложение подпрограмм, локализацию и ограничение области «видимости» данных. Среди наиболее известных языков этой группы стоит назвать PL/1, ALGOL-68, Pascal, C.

Одновременно со структурным программированием появилось огромное количество языков, базирующихся на других концепциях, но большинство из них не выдержало конкуренции. Какие-то языки были просто забыты, идеи других были в дальнейшем использованы в следующих версиях развиваемых языков.

Дальнейший рост сложности и размеров разрабатываемого программного обеспечения потребовал развития структурирования данных. Как следствие этого в языках появляется возможность определения пользовательских типов данных. Одновременно усилилось стремление разграничить доступ к глобальным данным программы, чтобы уменьшить

количество ошибок, возникающих при работе с глобальными данными. В результате появилась и начала развиваться технология модульного программирования.

Модульное программирование предполагает выделение групп подпрограмм, использующих одни и те же глобальные данные в отдельно компилируемые модули (библиотеки подпрограмм), например, модуль графических ресурсов, модуль подпрограмм вывода на принтер (рисунок 1.5). Связи между модулями при использовании данной технологии осуществляются через специальный интерфейс, в то время как доступ к реализации модуля (телам под- программ и некоторым «внутренним» переменным) запрещен. Эту технологию поддерживают современные версии языков Pascal и C (C++), языки Ада и Modula.

Использование модульного программирования существенно упростило разработку программного обеспечения несколькими программистами. Теперь каждый из них мог разрабатывать свои модули независимо, обеспечивая взаимодействие модулей через специально оговоренные межмодульные интерфейсы. Кроме того, модули в дальнейшем без изменений можно было использовать в других разработках, что повысило производительность труда программистов.

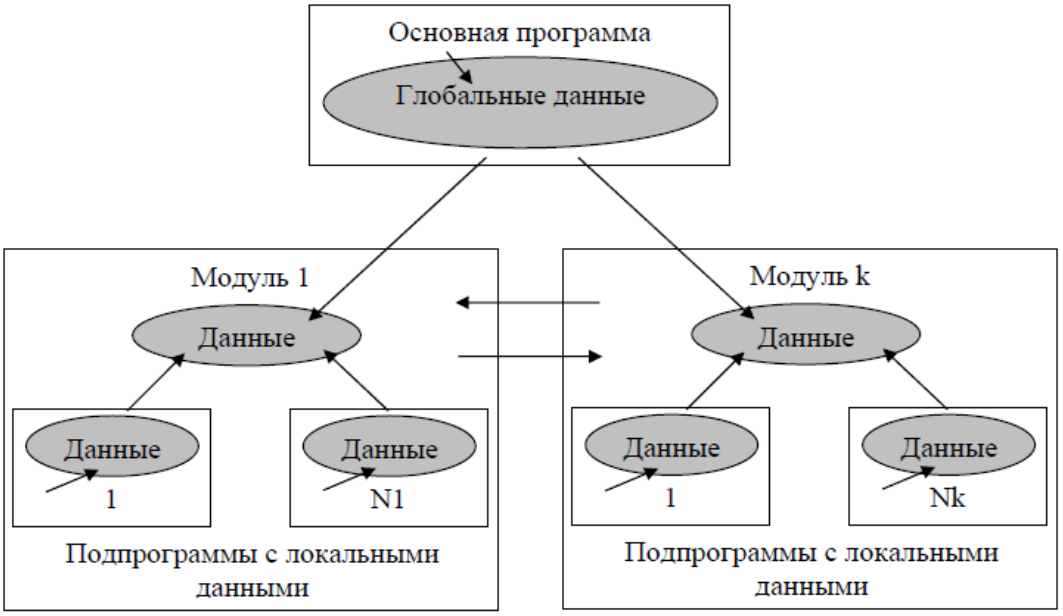


Рисунок 1.5 - Модульная структура программ

Практика показала, что структурный подход в сочетании с модульным программированием позволяет получать достаточно надежные программы, размер которых не превышает 100 000 операторов. Узким местом модульного программирования является то, что ошибка в интерфейсе при вызове подпрограммы выявляется только при выполнении программы (из-за раздельной компиляции модулей обнаружить эти ошибки раньше невозможно). При увеличении размера программы обычно возрастает сложность межмодульных интерфейсов, и с некоторого момента предусмотреть взаимовлияние отдельных частей программы становится практически невозможно. Для разработки программного обеспечения большого объема было предложено использовать объектный подход.

1.1.3 Этап 3. Объектный подход к программированию (с середины 1980-х гг. до нашего времени)

Объектно-ориентированное программирование определяется как технология создания сложного программного обеспечения, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса (рисунок 1.6), а классы образуют иерархию с наследованием свойств. Взаимодействие программных объектов в такой системе осуществляется путем передачи сообщений (рисунок 1.7).

Объектная структура программы впервые была использована в языке имитационного моделирования сложных систем Simula, появившемся еще в 60-х гг. XX в. Естественный для языков моделирования способ представления программы получил развитие в другом специализированном языке моделирования – языке Smalltalk (70-е гг. XX в.), а затем был использован в новых версиях универсальных языков программирования, таких как Pascal, C+

+, Modula, Java.

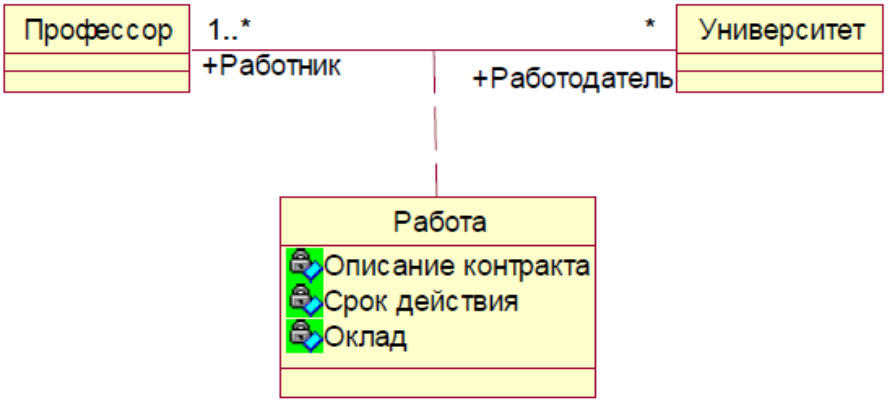


Рисунок 1.6 - Структура объектно-ориентированной программы в виде связанных классов

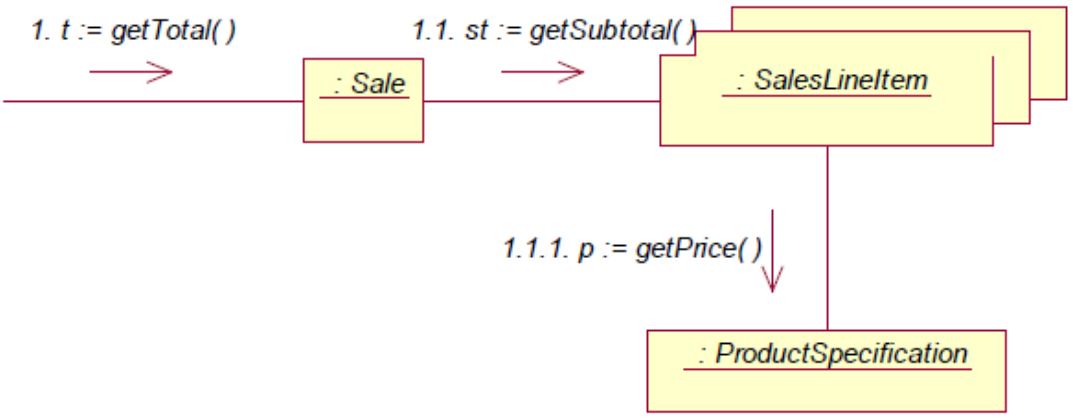


Рисунок 1.7 - Взаимодействие объектов в объектно-ориентированных программах

Основным достоинством объектно-ориентированного программирования по сравнению с модульным программированием является «более естественная» декомпозиция программного обеспечения, которая существенно облегчает его разработку. Это приводит к более полной локализации данных и интегрированию их с подпрограммами обработки, что позволяет вести практически независимую разработку отдельных частей (объектов) программы. Кроме этого, объектный подход предлагает новые способы организации программ, основанные на механизмах наследования, полиморфизма, композиции, наполнения. Эти механизмы позволяют конструировать сложные объекты из сравнительно простых. В результате существенно увеличивается показатель повторного использования кодов и появляется возможность создания библиотек классов для различных применений.

Бурное развитие технологий программирования, основанных на объектном подходе, позволило решить многие проблемы. Так, были созданы среды, поддерживающие визуальное программирование, например, Delphi, C++ Builder, Visual C++ и т. д. При использовании визуальной среды у программиста появляется возможность проектировать некоторую часть, например, интерфейсы будущего продукта, с применением визуальных средств добавления и настройки специальных библиотечных компонентов. Результатом визуального проектирования является заготовка будущей программы, в которую уже внесены соответствующие коды.

Использование объектного подхода имеет много преимуществ, однако его конкретная реализация в объектно-ориентированных языках программирования, таких как Pascal и C++, имеет существенные недостатки:

- фактически отсутствуют стандарты компоновки двоичных результатов компиляции объектов в единое целое даже в пределах одного языка программирования: компоновка объектов, полученных разными компиляторами C++ в лучшем случае проблематична, что приводит к необходимости разработки:
- программного обеспечения с использованием средств и возможностей одного языка программирования высокого уровня и одного компилятора, а значит, требует наличия

исходных кодов используемых библиотек классов;

- изменения реализации одного из программных объектов, как минимум, связано с перекомпиляцией соответствующего модуля и перекомпоновкой всего программного обеспечения, использующего данный объект.

Таким образом, при использовании этих языков программирования сохраняется зависимость модулей программного обеспечения от адресов экспортируемых полей и методов, а также структур и форматов данных. Эта зависимость объективна, так как модули должны взаимодействовать между собой, обращаясь к ресурсам друг друга. Связи модулей нельзя разорвать, но можно попробовать стандартизировать их взаимодействие, на чем и основан компонентный подход к программированию.

1.1.4 Этап 4. Компонентный подход и CASE-технологии (с середины 1990-х гг. до нашего времени)

Компонентный подход предполагает построение программного обеспечения из отдельных компонентов – физически отдельно существующих частей программного обеспечения, которые взаимодействуют между собой через стандартизованные двоичные интерфейсы. В отличие от обычных объектов объекты-компоненты можно собрать в динамически вызываемые библиотеки или исполняемые файлы, распространять в двоичном виде (без исходных текстов) и использовать в любом языке программирования, поддерживающем соответствующую технологию. Сегодня рынок объектов стал реальностью: так, в Интернете существуют узлы, предоставляющие большое количество компонентов, рекламой компонентов забиты журналы. Это позволяет программистам создавать продукты, хотя бы частично состоящие из повторно использованных частей, т. е. применять технологию, хорошо зарекомендовавшую себя в области проектирования аппаратуры.

Компонентный подход лежит в основе технологий, разработанных на базе COM (Component Object Model – компонентная модель объектов), и технологии создания распределенных приложений CORBA (Common Object Request Broker Architecture – общая архитектура с посредником обработки запросов объектов). Эти технологии используют сходные принципы и различаются лишь особенностями их реализации.

Технология COM фирмы Microsoft является развитием технологии OLE (Object Linking and Embedding – связывание и внедрение объектов), которая использовалась в ранних версиях Windows для создания составных документов. Технология COM определяет общую парадигму взаимодействия программ любых типов: библиотек, приложений, операционной системы, т. е. позволяет одной части программного обеспечения использовать функции (*службы*), предоставляемые другой, независимо от того, функционируют ли эти части в пределах одного процесса, в разных процессах на одном компьютере или на разных компьютерах (рисунок 1.8). Модификация COM, обеспечивающая передачу вызовов между компьютерами, называется DCOM (Distri- buted COM – распределенная COM).

По технологии COM приложение предоставляет свои службы, используя специальные объекты – объекты COM, которые являются экземплярами классов COM. Объект COM, так же как обычный объект, включает поля и методы, но в отличие от обычных объектов каждый объект COM может реализовывать несколько интерфейсов, обеспечивающих доступ к его полям и функциям. Это достигается за счет организации отдельной таблицы адресов методов для каждого интерфейса (по типу таблиц виртуальных методов).

При этом интерфейс обычно объединяет несколько однотипных функций.

Кроме того, классы COM поддерживают наследование интерфейсов, но не поддерживают наследования реализации, т. е. не наследуют код методов, хотя при необходимости объект класса-потомка может вызвать метод родителя.

Каждый интерфейс имеет имя, начинающееся с символа I и глобальный уникальный идентификатор IID (Interface IDentifier). Любой объект COM обязательно реализует интерфейс (Unknown (на схемах этот интерфейс всегда располагают сверху). Использование этого интерфейса позволяет получить доступ к остальным интерфейсам объекта.

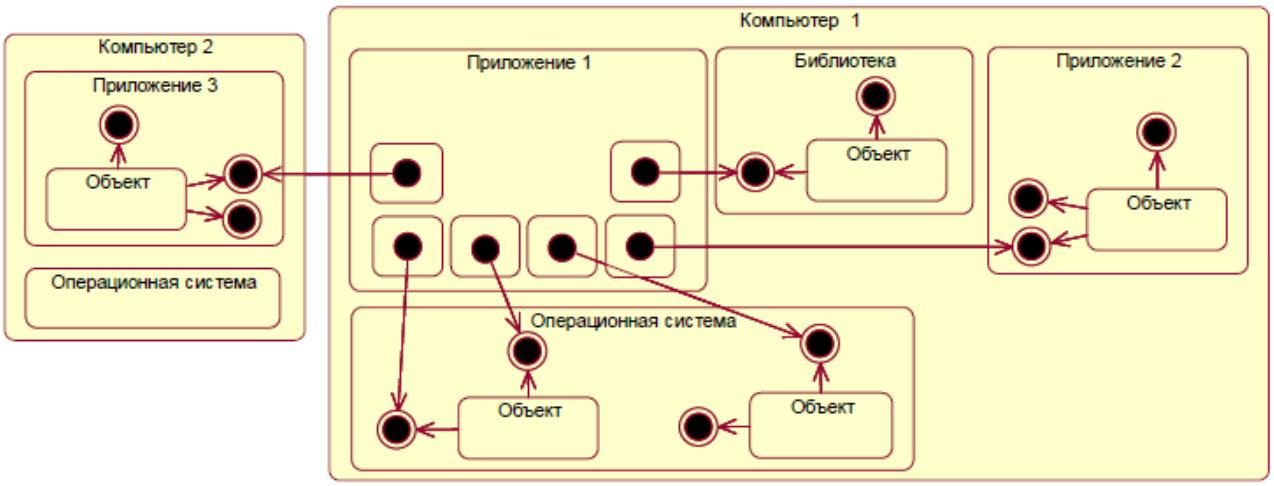


Рисунок 1.8 - Взаимодействие программных компонентов различных типов

Объект всегда функционирует в составе сервера – динамической библиотеки или исполняемого файла, которые обеспечивают функционирование объекта. Различают три типа серверов:

- внутренний сервер: реализуется динамическими библиотеками, которые подключаются к приложению-клиенту и работают в одном с ними адресном пространстве. Это наиболее эффективный сервер, кроме того, он не требует специальных средств;
- локальный сервер: создается отдельным процессом (exe), который работает на одном компьютере с клиентом;
- удаленный сервер: создается процессом, который работает на другом компьютере.

Например, Microsoft Word является локальным сервером. Он включает множество объектов, которые могут использоваться другими приложениями.

Взаимодействие клиента и сервера обеспечивается базовыми механизмами COM или DCOM, поэтому клиенту безразлично местонахождение объекта. При использовании локальных и удаленных серверов в адресном пространстве клиента создается *proxy-объект* – заместитель объекта COM, а в адресном пространстве сервера COM – *заглушка,* соответствующая клиенту. Получив задание от клиента, заместитель упаковывает его параметры и, используя службы операционной системы, передает вызов заглушке. Заглушка распаковывает задание и передает его объекту COM. Результат возвращается клиенту в обратном порядке.

1.1.5 этап 5. Разработка, ориентированная на архитектуру и CASE-технологии (с начала XXI в. до нашего времени)

До конца XX в. все возможные проектные модели употребляли исключительно для документирования промежуточных и заключительных этапов разработки программного обеспечения, для чего применялись различные графические нотации и технологии, которые впоследствии были использованы для создания стандарта объектного моделирования.

В ноябре 1997 г. после продолжительного процесса объединения различных методик группа OMG (Object Management Group) приняла получившийся в результате унифицированный язык моделирования (Unified Model Language – UML) в качестве стандарта. В 2001 г. члены OMG начали работу над новой версией UML, добавляя в нее недостающие элементы и устраняя недостатки, выявленные в UML1. Версия UML2 была принята в 2004 г. С официальной спецификацией UML можно ознакомиться на веб-сайте OMG по адресу www.omg.org.

Идея создания языка UML включала в себя не только реализацию стандарта для документирования и общения разработчиков, но и реализацию возможности использования UML как языка программирования. Этот момент вызывал массу проблем при осуществлении, так как язык визуального моделирования по определению не мог содержать в себе всей выразительности объектно-ориентированных языков в плане проектирования (программирования) динамики и реализации алгоритмов. Нужен был язык, который на более высоком (абстрактном) уровне смог бы обеспечить разработку на UML. Такой язык был создан – это объектный язык ограничений OCL (Object Con- straint Language).

Тенденции развития средств разработки программных систем заключаются в создании

таких средств, которые обеспечили бы не только автоматизацию всех этапов и процессов разработки программных систем, но и связь между результатами этапов. Одним из ключевых соединительных узлов является связь между проектными моделями и программным кодом. Когда разработка программных систем начинается от проектирования ее структуры до последующего кодирования и все изменения в функциях разрабатываемой системы реализуются, начиная с перепроектирования архитектуры, то такая технология называется ориентированной на архитектуру (Model Driven Ar- chitecture – MDA).

Компания Borland, начиная с седьмой версии своей среды разработки (Delphi) уже использует набор компонент, реализующий подход, ориентированный на архитектуру, но в этой версии присутствует еще масса недостатков и недоработок. В последней своей версии (Delphi 2006) компания Borland модернизировала технологию, ориентированную на архитектуру, что позволило использовать ее для разработки промышленных приложений, в том числе и для платформы Microsoft Framework .Net.

1.2 Эволюция моделей жизненного цикла программного обеспечения

1.2.1 Каскадная модель

Первоначально (1970–1985 гг.) была предложена и использовалась *каскадная схема разработки программного обеспечения* (ПО) (рисунок 1.9), которая предполагала, что переход на следующую стадию осуществляется после того, как полностью будут завершены проектные операции предыдущей стадии и получены все исходные данные для следующей стадии.

Именно такую схему и используют обычно при блочно-иерархическом подходе к разработке сложных *технических* объектов, обеспечивая очень высокие параметры эффективности разработки. Однако данная схема оказалась применимой только к созданию систем, для которых в самом начале разработки удавалось точно и полно сформулировать все требования. Это уменьшало вероятность возникновения в процессе разработки проблем, связанных с принятием неудачного решения на предыдущих стадиях. На практике такие разработки встречаются крайне редко.

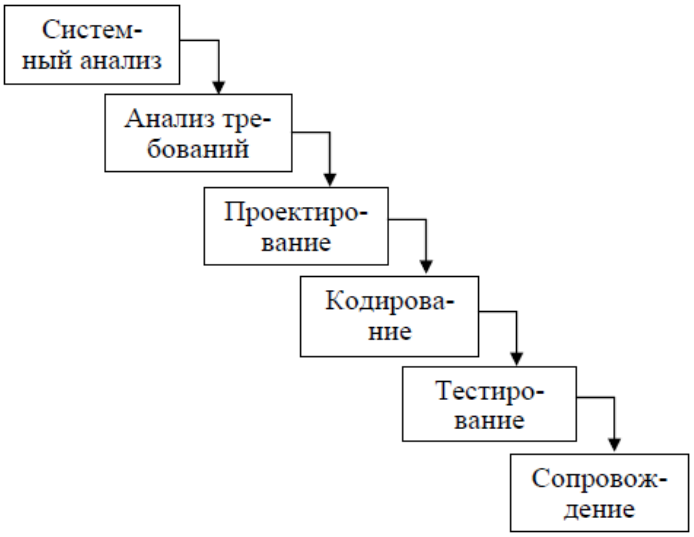


Рисунок - 1.9 Каскадная модель разработки программного обеспечения

Схема, поддерживающая итерационный характер процесса разработки, была названа схемой с промежуточным контролем (рисунок 1.10). Контроль, который выполняется по данной схеме после завершения каждого этапа, позволяет при необходимости вернуться на любой уровень и внести необходимые изменения.

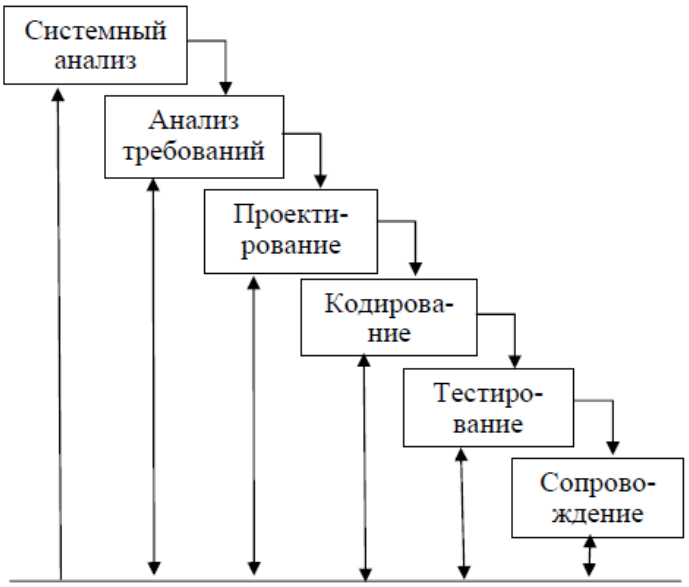


Рисунок 1.10 - Каскадная модель разработки ПО с промежуточным контролем

Охарактеризуем содержание основных этапов.

Подразумевается, что разработка начинается на системном уровне и проходит через анализ, проектирование, кодирование, тестирование и сопровождение. При этом моделируются действия стандартного инженерного цикла.

Системный анализ задает роль каждого элемента в компьютерной системе, взаимодействие элементов друг с другом. Поскольку ПО является лишь частью большой системы, то анализ начинается с определения требований ко всем системным элементам и назначения подмножества этих требований программному элементу. Необходимость системного подхода явно проявляется, когда формируется интерфейс ПО с другими элементами (аппаратурой, людьми, базами данных). На этом же этапе начинается решение задачи планирования проекта ПО. В ходе планирования проекта определяются объем проектных работ и их риск, необходимые трудозатраты, формируются рабочие задачи и план-график работ.

Анализ требований относится к программному элементу – программному обеспечению. Уточняются и детализируются его функции, характеристики и интерфейс.

Все определения документируются в *спецификации анализа.* Здесь же завершается решение задачи планирования проекта.

Проектирование состоит в создании представлений:

- архитектуры ПО;
- модульной структуры ПО;
- алгоритмической структуры ПО;
- структуры данных;
- входного и выходного интерфейса (входных и выходных форм данных).

Исходные данные для проектирования содержатся в *спецификации анализа,* т. е. в ходе проектирования выполняется трансляция требований к ПО во множество проектных представлений: при решении задач проектирования основное внимание уделяется качеству будущего программного продукта.

Кодирование – перевод результатов проектирования в текст на языке программирования.

Тестирование – выполнение программы для выявления дефектов в функциях, логике и форме реализации программного продукта.

Сопровождение – это внесение изменений в эксплуатируемое ПО. Цели изменений:

- исправление ошибок;
- адаптация к изменениям внешней для ПО среды;
- усовершенствование ПО по требованиям заказчика.

Сопровождение ПО состоит в повторном применении каждого из предшествующих шагов (этапов) жизненного цикла к существующей программе, но не в разработке новой программы.

Как и любая инженерная схема, классический жизненный цикл имеет достоинства и

недостатки.

Достоинства классического жизненного цикла:

- получение в конце каждой стадии законченного набора проектной документации, отвечающего требованиям полноты и согласованности;
- простота планирования процесса разработки.

Недостатки классического жизненного цикла:

- реальные проекты часто требуют отклонения от стандартной последовательности шагов;
- цикл основан на точной формулировке исходных требований к ПО (реально в начале проекта требования заказчика определены лишь частично);
- результаты проекта доступны заказчику только в конце работы.

1.2.2 Спиральная модель

Спиральная модель (автор Барри Боэм, 1988) базируется на лучших свойствах классического жизненного цикла и макетирования, к которым добавляется новый элемент – анализ риска, отсутствующий в этих парадигмах.

Как показано на рисунке 1.11, модель определяет четыре действия, представляемые четырьмя квадрантами спирали.

1. Планирование – определение целей, вариантов и ограничений.
2. Анализ риска – анализ вариантов и распознавание/выбор риска.
3. Конструирование – разработка продукта следующего уровня.
4. Оценивание – оценка заказчиком текущих результатов конструирования.

Интегрирующий аспект спиральной модели очевиден при учете радиального измерения спирали. С каждой итерацией по спирали (продвижением от центра к периферии) строятся все более полные версии ПО.

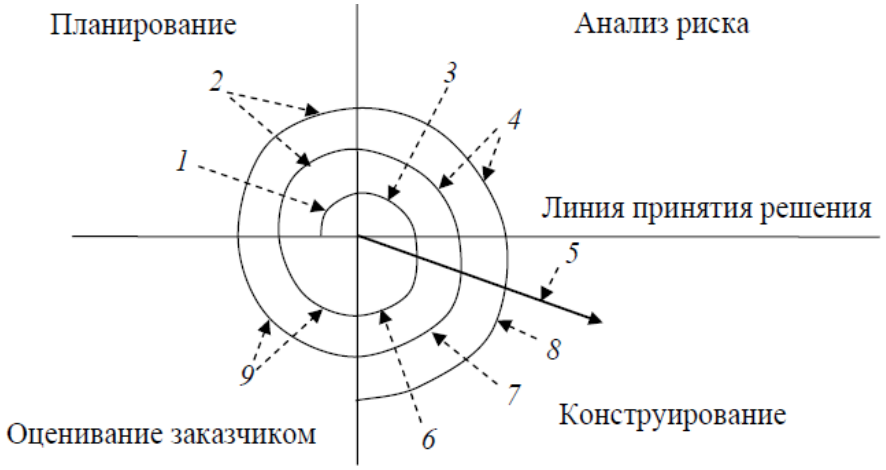


Рисунок 1.11 - Спиральная модель: *1* – начальный сбор требований и планирование проекта; *2* – та же работа, но на основе рекомендаций заказчика; *3* – анализ риска на основе начальных требований; *4* – анализ риска на основе реакции заказчика; *5* – переход к комплексной системе; *6* – начальный макет системы; *7* – следующий уровень макета; *8* – сконструированная система; *9* – оценивание заказчиком

В первом витке спирали определяются начальные цели, варианты и ограничения, распознается и анализируется риск. Если анализ риска показывает неопределенность требований, на помощь разработчику и заказчику приходит макетирование (используемое в квадранте конструирования). Для дальнейшего определения проблемных и уточненных требований может быть использовано моделирование. Заказчик оценивает инженерную (конструкторскую) работу и вносит предложения по модификации (квадрант оценки заказчиком). Следующая фаза планирования и анализа риска базируется на предложениях заказчика. В каждом цикле по спирали результаты анализа риска формируются в виде «продолжать, не продолжать». Если риск слишком велик, проект может быть остановлен.

В большинстве случаев движение по спирали продолжается, с каждым шагом продвигая

разработчиков к более общей модели системы. В каждом цикле по спирали требуется конструирование (нижний правый квадрант), которое может быть реализовано классическим жизненным циклом или макетированием. Заметим, что количество действий по разработке (происходящих в правом нижнем квадранте) возрастает по мере продвижения от центра спирали.

Достоинства спиральной модели:

- наиболее реально (в виде эволюции) отображает разработку программного обеспечения;
- позволяет явно учитывать риск на каждом витке эволюции разработки;
- включает шаг системного подхода в итерационную структуру разработки;
- использует моделирование для уменьшения риска и совершенствования программного изделия.

Недостатки спиральной модели:

- повышенные требования к заказчику;
- трудности контроля и управления временем разработки.

1.2.3 Макетирование

Достаточно часто заказчик не может сформулировать подробные требования по вводу, обработке или выводу данных для будущего программного продукта. С другой стороны, разработчик может сомневаться в приспосабливаемости продукта под операционную систему, в форме диалога с пользователем или в эффективности реализуемого алгоритма. В этих случаях целесообразно использовать макетирование.

Основная цель макетирования – снять неопределенности в требованиях заказчика.

Макетирование (прототипирование) – это процесс создания модели требуемого программного продукта.

Модель может принимать одну из трех форм:

- бумажный макет или макет на основе ПК (изображает или рисует человеко-машинный диалог);
- работающий макет (выполняет некоторую часть требуемых функций);
- существующая программа (характеристики которой затем должны быть улучшены).

Макетирование основывается на многократном повторении итераций, в которых участвуют заказчик и разработчик (рисунок 1.12).

Рисунок 1.12 - Макетирование

Последовательность действий при макетировании представлена на рисунке 1.13. Макетирование начинается со сбора и уточнения требований к создаваемому ПО. Разработчик и заказчик встречаются и определяют все цели ПО, устанавливают, какие требования известны, а какие предстоит доопределить.

Затем выполняется быстрое проектирование. В нем внимание сосредоточивается на тех характеристиках ПО, которые должны быть видимы пользователю.

Быстрое проектирование приводит к построению макета.

Рисунок 1.13 - Последовательность действий при макетировании

Макет оценивается заказчиком и используется для уточнения требований к ПО. Итерации повторяются до тех пор, пока макет не выявит все требования заказчика и тем самым не даст возможность разработчику понять, что должно быть сделано.

Достоинство макетирования – обеспечивает определение полных требований к ПО.

Недостатки макетирования:

- заказчик может принять макет за продукт;
- разработчик может принять макет за продукт.

Поясним суть недостатков. Когда заказчик видит работающую версию ПО, он перестает сознавать, что детали макета скреплены «жевательной резинкой и проволокой»; он забывает, что в погоне за работающим вариантом оставлены нерешенными вопросы качества и удобства сопровождения ПО. Когда заказчику говорят, что продукт должен быть перестроен, он начинает возмущаться и требовать, чтобы макет «в три приема» был превращен в рабочий продукт. Очень часто это отрицательно сказывается на управлении разработкой ПО. С другой стороны, для быстрого получения работающего макета разра- ботчик часто идет на определенные компромиссы. Могут использоваться не самые подходящие язык программирования или операционная система. Для простой демонстрации возможностей может применяться неэффективный алгоритм. Спустя некоторое время разработчик забывает о причинах, по которым эти средства не подходят. В результате далеко не идеальный выбранный вариант интегрируется в систему.

Очевидно, что преодоление этих недостатков требует борьбы с житейским соблазном – принять желаемое за действительное.

1.2.4 Быстрая разработка приложений

Модель быстрой разработки приложений (Rapid Application Development) обеспечивает экстремально короткий цикл разработки. RAD – высокоскоростная адаптация линейной последовательной модели, в которой быстрая разработка достигается за счет использования компонентно-ориентированного конструирования. Если требования полностью определены, а проектная область ограничена, RAD-процесс позволяет группе создать полностью функциональную систему за очень короткое время (60–90 дней).

RAD-подход ориентирован на разработку информационных систем и выделяет

следующие этапы:

1. Бизнес-моделирование. Моделируется информационный поток между бизнес-функциями. Ищутся ответы на следующие вопросы: Какая информация руководит бизнес-процессом? Какая информация генерируется? Кто генерирует ее? Где информация применяется? Кто обрабатывает ее?

2. Моделирование данных. Информационный поток, определенный на этапе бизнес-моделирования, отображается в наборе объектов данных, которые требуются для поддержки бизнеса. Идентифицируются характеристики (свойства, атрибуты) каждого объекта, определяются отношения между объектами;

3. Моделирование обработки. Определяются преобразования объектов данных, обеспечивающие реализацию бизнес-функций. Создаются описания обработки для добавления, модификации, удаления или нахождения (испраления) объектов данных;

4. Генерация приложения. Предполагается использование методов, ориентированных на языки программирования 4-го поколения. Вместо создания ПО с помощью языков программирования 3-го поколения RAD-процесс работает с повторно используемыми программными компонентами или создает повторно используемые компоненты. Для обеспечения конструирования применяются утилиты автоматизации;

5. Тестирование и объединение. Поскольку применяются повторно используемые компоненты, многие программные элементы уже протестированы. Это уменьшает время тестирования (хотя все новые элементы должны быть протестированы).

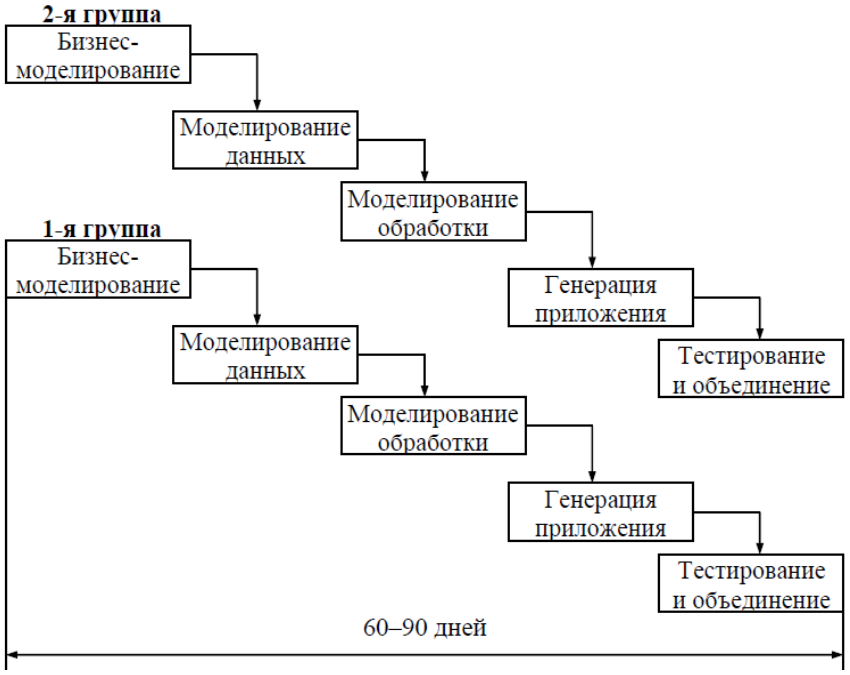


Рисунок 1.14 - Модель быстрой разработки приложений

Применение RAD возможно в том случае, когда каждая главная функция может быть завершена за 3 месяца. Каждая главная функция адресуется отдельной группе разработчиков, а затем интегрируется в целую систему.

Применение RAD имеет и свои недостатки, и ограничения:

- для больших проектов в RAD требуются существенные людские ресурсы (необходимо создать достаточное количество групп);
- RAD применима только для таких приложений, которые могут декомпозироваться на отдельные модули и в которых производительность не является критической величиной;
- RAD неприменима в условиях высоких технических рисков (т. е. при использовании новой технологии).

1.2.5 Компонентно-ориентированная модель

Компонентно-ориентированная модель является развитием спиральной. В этой модели конкретизируется содержание квадранта конструирования – оно отражает тот факт, что в современных условиях новая разработка должна основываться на повторном использовании существующих программных компонентов (рисунок 1.15).

Программные компоненты, созданные в реализованных программных проектах, хранятся в библиотеке. В новом программном проекте, исходя из требований заказчика, выявляются кандидаты в компоненты. Далее проверяется наличие этих кандидатов в библиотеке. Если они найдены, то компоненты извлекаются из библиотеки и используются повторно. В противном случае создаются новые компоненты, они применяются в проекте и включаются в библиотеку.

Достоинства компонентно-ориентированной модели:

- уменьшает на 30 % время разработки программного продукта;
- уменьшает стоимость программной разработки до 70 %;
- увеличивает в полтора раза производительность разработки.

Недостатком такой модели является сложность организации процесса разработки ПО по данной модели.

1.2.6 XP-процесс

Экстремальное программирование (eXtreme Programming, XP) – облегченный (подвижный) процесс (или методология), главный автор которого Кент Бек (1999). XP-процесс ориентирован на группы малого и среднего размера, строящие программное обеспечение в условиях неопределенных или быстро изменяющихся требований. XP-группу образуют до 10 сотрудников, которые размещаются в одном помещении.

Основная идея XP – устранить высокую стоимость изменения, характерную для приложений с использованием объектов, паттернов и реляционных баз данных. Поэтому XP-процесс должен быть высокодинамичным процессом. XP-группа имеет дело с изменениями требований на всем протяжении итерационного цикла разработки, причем цикл состоит из очень коротких итераций. Четырьмя базовыми действиями в XP-цикле являются: кодирование, тестирование, выслушивание заказчика и проектирование. Динамизм обеспечивается с помощью четырех характеристик: непрерывной связи с заказчиком (и в пределах группы), простоты (всегда выбирается минимальное решение), быстрой обратной связи (с помощью модульного и функционального тестирования), смелости в проведении профилактики возможных проблем.

Большинство принципов, поддерживаемых в XP (минимальность, простота, эволюционный цикл разработки, малая длительность итерации, участие пользователя, оптимальные стандарты кодирования и т. д.), продиктованы здравым смыслом и применяются в любом упорядоченном процессе. Просто в XP эти принципы достигают «экстремальных значений» (таблица 1.1).

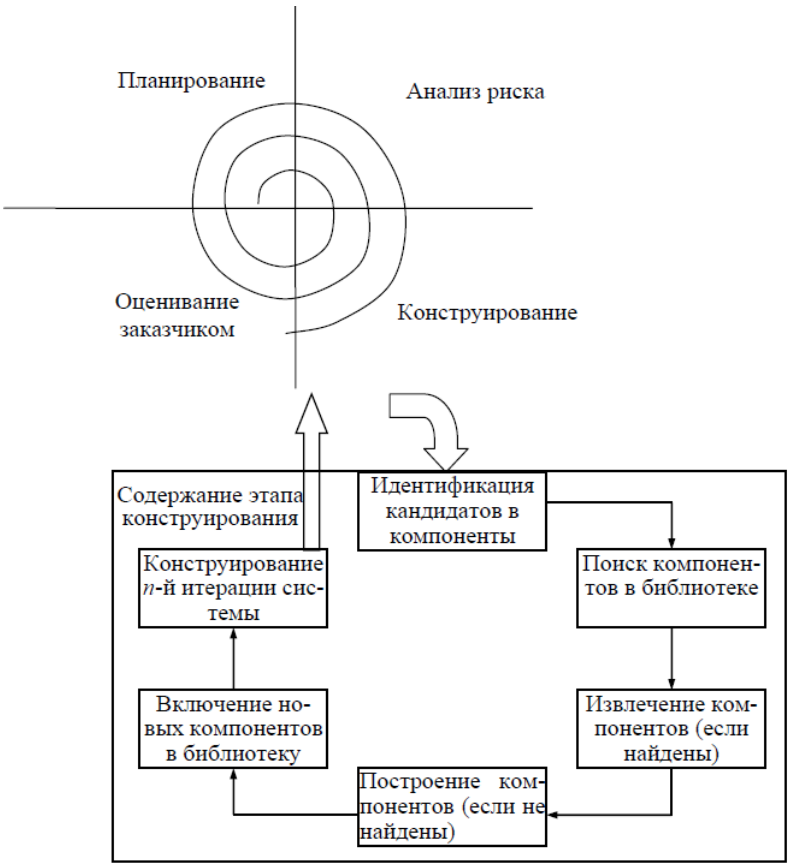


Рисунок 1.15 - Компонентно-ориентированная модель

Таблица 1.1 Экстремумы в экстремальном программировании

Практика здравого смысла	XP-экстремум	XP-реализация
Проверки кода	Код проверяется все время	Парное программирование
Тестирование	Тестирование выполняется все время, даже с помощью заказчиков	Тестирование модуля, функциональное тестирование
Проектирование	Проектирование является частью ежедневной деятельности каждого разработчика	Реорганизация (refactoring)
Простота	Для системы выбирается простейшее проектное решение, поддерживающее ее текущую функциональность	Самая простая вещь, которая могла бы работать
Архитектура	Каждый постоянно работает над уточнением архитектуры.	Метафора
Тестирование интеграции	Интегрируется и тестируется несколько раз в день	Непрерывная интеграция
Короткие итерации	Итерации являются предельно короткими, продолжаются секунды, минуты, часы, а не недели, месяцы или годы	Игра планирования

Тот, кто принимает принцип минимального решения за хакерство, ошибается; в действительности XP – строго упорядоченный процесс. Простые решения, имеющие высший приоритет, в настоящее время рассматриваются как наиболее ценные части системы, в отличие

от проектных решений, которые пока не нужны, а могут (в условиях изменения требований и операционной среды) и вообще не понадобиться. Базис XP образуют перечисленные ниже двенадцать методов.

1. Игра планирования (Planning game) – быстрое определение области действия следующей реализации путем объединения деловых приоритетов и технических оценок. Заказчик формирует область действия, приоритетность и сроки с точки зрения бизнеса, а разработчики оценивают и прослеживают продвижение (прогресс).

2. Частая смена версий (Small releases) – быстрый запуск в производство простой системы. Новые версии реализуются в очень коротком (двухнедельном) цикле.

3. Метафора (Metaphor) – вся разработка проводится на основе простой, общедоступной истории о том, как работает вся система.

4. Простое проектирование (Simple design) – проектирование выполняется настолько просто, насколько это возможно в данный момент.

5. Тестирование (Testing) – непрерывное написание тестов для модулей, которые должны выполняться безупречно; заказчики пишут тесты для демонстрации законченности функций. «Тестируй, а затем кодируй» означает, что входным критерием для написания кода является «отказавший» тестовый вариант.

6. Реорганизация (Refactoring) – система реструктурируется, но ее поведение не изменяется; цель – устранить дублирование, улучшить взаимодействие, упростить систему или добавить в нее гибкость.

7. Парное программирование (Pair programming) – весь код пишется двумя программистами, работающими на одном компьютере.

8. Коллективное владение кодом (Collective ownership) – любой разработчик может улучшать любой код системы в любое время.

9. Непрерывная интеграция (Continuous integration) – система интегрируется и строится много раз в день по мере завершения каждой задачи. Непрерывное регрессионное тестирование, т. е. повторение предыдущих тестов, гарантирует, что изменения требований не приведут к регрессу функциональности.

10. 40-часовая неделя (40-hour week) – как правило, работают не более 40 часов в неделю. Нельзя удваивать рабочую неделю за счет сверхурочных работ.

11. Локальный заказчик (On-site customer) – в группе все время должен находиться представитель заказчика, действительно готовый отвечать на вопросы разработчиков.

12. Стандарты кодирования (Coding standards) – должны выдерживаться правила, обеспечивающие одинаковое представление программного кода во всех частях программной системы.

1.3 Стандарты, регламентирующие процесс разработки программного обеспечения

1.3.1 ГОСТ Р ИСО 9000–2001. Системы менеджмента качества. Основные положения и словарь

1.3.1.1 Предисловие

Стандарт разработан Всероссийским научно-исследовательским институтом сертификации (ВНИИС). Представляет собой аутентичный текст международного стандарта ИСО 9000–2000 «Системы менеджмента качества. Основные положения и словарь».

1.3.1.2 Введение

Семейство стандартов ИСО 9000, перечисленных ниже, было разработано для того, чтобы помочь организациям всех видов и размеров внедрять и обеспечивать функционирование эффективных систем менеджмента качества: ГОСТ Р ИСО 9000–2001 описывает основные положения систем менеджмента качества и устанавливает терминологию для систем менеджмента качества;

ГОСТ Р ИСО 9001–2001 определяет требования к системам менеджмента качества для тех случаев, когда организации необходимо продемонстрировать свою способность предоставлять продукцию, отвечающую требованиям потребителей и установленным к ней

обязательным требованиям. Направлен на повышение удовлетворенности потребителей;

ГОСТ Р ИСО 9004–2001 содержит рекомендации, рассматривающие как результативность, так и эффективность системы менеджмента качества. Целью этого стандарта является улучшение деятельности организации и удовлетворенность потребителей и других заинтересованных сторон;

ИСО 19011* содержит методические указания по аудиту (проверке) систем менеджмента качества и охраны окружающей среды.

Вместе они образуют согласованный комплекс стандартов на системы менеджмента качества, содействующий взаимопониманию в национальной и международной торговле.

Принципы менеджмента качества. Для успешного руководства организацией и ее функционирования необходимо направлять ее и управлять систематически и прозрачным способом. Успех может быть достигнут в результате внедрения и поддержания в рабочем состоянии системы менеджмента качества, разработанной для постоянного улучшения деятельности с учетом потребностей всех заинтересованных сторон. Управление организацией включает менеджмент качества наряду с другими аспектами менеджмента.

Восемь принципов менеджмента качества были определены для того, чтобы высшее руководство могло следовать им с целью улучшения деятельности организации.

1. Ориентация на потребителя

Организации зависят от своих потребителей, и поэтому должны понимать их текущие и будущие потребности, выполнять их требования и стремиться превзойти их ожидания.

2. Лидерство руководителя

Руководители обеспечивают единство цели и направления деятельности организации. Они должны создавать и поддерживать внутреннюю среду, в которой работники могут быть полностью вовлечены в решение задач организации.

3. Вовлечение работников

Работники всех уровней составляют основу организации, и их полное вовлечение дает возможность организации с выгодой использовать их способности.

4. Процессный подход

Желаемый результат достигается эффективнее, когда деятельностью и соответствующими ресурсами управляют как процессом.

5. Системный подход к менеджменту

Выявление, понимание и менеджмент взаимосвязанных процессов как системы содействуют результативности и эффективности организации при достижении ее целей.

6. Постоянное улучшение

Постоянное улучшение деятельности организации в целом следует рассматривать как ее неизменную цель.

7. Принятие решений, основанное на фактах

Эффективные решения основываются на анализе данных и информации.

8. Взаимовыгодные отношения с поставщиками

Организация и ее поставщики взаимозависимы, и отношения взаимной выгоды повышают способность обеих сторон создавать ценности.

Эти восемь принципов менеджмента качества образуют основу для стандартов на системы менеджмента качества, входящих в семейство ИСО 9000.

1.3.1.3 Область применения

Настоящий стандарт устанавливает основные положения систем менеджмента качества, являющихся объектом стандартов семейства ИСО 9000, и определяет соответствующие термины.

Настоящий стандарт может использоваться:

- организациями, стремящимися добиться преимущества посредством внедрения системы менеджмента качества;
- организациями, стремящимися получить уверенность в том, что их заданные требования к продукции будут выполнены поставщиками;
- пользователями продукции;
- теми, кто заинтересован в едином понимании терминологии, применяемой в менеджменте качества (например, поставщики, потребители, регламентирующие органы);

- теми сторонами, внутренними или внешними по отношению к организации, которые оценивают систему менеджмента качества или проверяют ее на соответствие требованиям ГОСТ Р ИСО 9001 (например, аудиторы, органы по сертификации/регистрации);

- теми сторонами, внутренними или внешними по отношению к организации, которые консультируют или проводят обучение по системе менеджмента качества, соответствующей данной организации;

- разработчиками соответствующих стандартов.

1.3.1.4 Основные положения систем менеджмента качества

Обоснование необходимости систем менеджмента качества. Системы менеджмента качества могут содействовать организациям в повышении удовлетворенности потребителей.

Потребителям необходима продукция, характеристики которой удовлетворяли бы их потребности и ожидания. Эти потребности и ожидания, как правило, отражаются в технических условиях на продукцию и обычно считаются требованиями потребителей. Требования могут быть установлены потребителем в контракте или определены самой организацией. В любом случае приемлемость продукции в конечном счете устанавливает потребитель. Поскольку потребности и ожидания потребителей меняются, организации также испытывают давление, обусловленное конкуренцией и техническим прогрессом, они должны постоянно совершенствовать свою продукцию и свои процессы.

Системный подход к менеджменту качества побуждает организации анализировать требования потребителей, определять процессы, способствующие получению продукции, приемлемой для потребителей, а также поддерживать эти процессы в управляемом состоянии. Система менеджмента качества может быть основой постоянного улучшения с целью увеличения вероятности повышения удовлетворенности как потребителей, так и других заинтересованных сторон. Она дает уверенность самой организации и потребителям в ее способности поставлять продукцию, полностью соответствующую требованиям.

Требования к системам менеджмента качества и требования к продукции. Семейство стандартов ИСО 9000 проводит различие между требованиями к системам менеджмента качества и требованиями к продукции.

Требования к системам менеджмента качества установлены в ГОСТ Р ИСО 9001. Они являются общими и применимыми к организациям в любых секторах промышленности или экономики независимо от категории продукции. ГОСТ Р ИСО 9001 не устанавливает требований к продукции.

Требования к продукции могут быть установлены потребителями или организацией, исходя из предполагаемых запросов потребителей или требований регламентов. Требования к продукции и в ряде случаев к связанным с ней процессам могут содержаться, например в технических условиях, стандартах на продукцию, стандартах на процессы, контрактных соглашениях и регламентах.

Подход к системам менеджмента качества. Подход к разработке и внедрению системы менеджмента качества состоит из нескольких ступеней, включающих:

- установление потребностей и ожиданий потребителей и других заинтересованных сторон;

- разработку политики и целей организации в области качества;

- установление процессов и ответственности, необходимых для достижения целей в области качества;

- установление и определение необходимых ресурсов и обеспечение ими для достижения целей в области качества;

- разработку методов для измерения результативности и эффективности каждого процесса;

- применение данных этих измерений для определения результативности и эффективности каждого процесса;

- определение средств, необходимых для предупреждения несоответствий и устранения их причин;

- разработку и применение процесса для постоянного улучшения системы менеджмента качества.

Такой подход также применяется для поддержания в рабочем состоянии и улучшения имеющейся системы менеджмента качества.

Организация, принимающая указанный выше подход, создает уверенность в возможностях своих процессов и качестве своей продукции, а также обеспечивает основу для постоянного улучшения. Это может привести к возрастанию удовлетворенности потребителей и других заинтересованных сторон и успеху организации.

Процессный подход. Любая деятельность или комплекс деятельности, в которой используются ресурсы для преобразования входов в выходы, может рассматриваться как процесс.

Чтобы результативно функционировать, организации должны определять и управлять многочисленными взаимосвязанными и взаимодействующими процессами. Часто выход одного процесса образует непосредственно вход следующего. Систематическая идентификация и менеджмент применяемых организацией процессов и прежде всего обеспечения их взаимодействия могут считаться «процессным подходом».

Назначение настоящего стандарта – побуждать принятие процессного подхода к менеджменту организации.

На рисунке 1.16 приведена основанная на процессном подходе система менеджмента качества, описанная в семействе стандартов ИСО 9000. Он показывает, что заинтересованные стороны играют существенную роль в предоставлении организации входных данных. Наблюдение за удовлетворенностью заинтересованных сторон требует оценки информации, касающейся восприятия заинтересованными сторонами степени выполнения их потребностей и ожиданий. Модель (рисунок 1.16), не показывает процессы на детальном уровне.

Рисунок 1.16 - Модель системы менеджмента качества, основанной на процессном подходе

Политика и цели в области качества. Политика и цели в области качества устанавливаются, чтобы служить ориентиром для организации. Они определяют желаемые результаты и способствуют использованию организацией ресурсов для достижения этих результатов. Политика в области качества обеспечивает основу для разработки и анализа целей в области качества. Цели в области качества необходимо согласовывать с политикой в области качества и приверженностью к постоянному улучшению, а результаты должны быть измеримыми.

Достижение целей в области качества может оказывать позитивное воздействие на качество продукции, эффективность работы и финансовые показатели и, следовательно, на удовлетворенность и уверенность заинтересованных сторон.

Роль высшего руководства в системе менеджмента качества. С помощью лидерства и реальных действий высшее руководство может создавать обстановку, способствующую полному вовлечению работников и эффективной работе системы менеджмента качества. Принципы менеджмента качества могут использоваться высшим руководством как основа для

выполнения своей роли в:

- разработке и поддержании политики и целей организации в области качества;
- популяризации политики и целей в области качества во всей организации для повышения осознания, мотивации и вовлечения персонала;
- обеспечении ориентации на требования потребителей во всей организации;
- обеспечении внедрения соответствующих процессов, позволяющих выполнять требования потребителей и других заинтересованных сторон и достигать целей в области качества;
- обеспечении разработки, внедрения и поддержания в рабочем состоянии эффективной системы менеджмента качества для достижения этих целей в области качества;
- обеспечении необходимыми ресурсами;
- проведении периодического анализа системы менеджмента качества;
- принятии решений в отношении политики и целей в области качества;
- принятии решений по мерам улучшения системы менеджмента качества.

Документация. Документация дает возможность передать смысл и последовательность действий. Ее применение способствует:

- достижению соответствия требованиям потребителя и улучшению качества;
- обеспечению соответствующей подготовки кадров;
- повторяемости и прослеживаемости;
- обеспечению объективных свидетельств;
- оцениванию эффективности и постоянной пригодности системы менеджмента качества.

Разработка документации не должна быть самоцелью, а должна добавлять ценность.

В системах менеджмента качества применяются следующие виды документов:

- документы, предоставляющие согласованную информацию о системе менеджмента качества организации, предназначенную как для внутреннего, так и внешнего пользования; к таким документам относятся руководства по качеству;
- документы, описывающие, как система менеджмента качества применяется к конкретной продукции, проекту или контракту; к таким документам относятся планы качества;
- документы, устанавливающие требования; к ним относятся документы, содержащие технические требования;
- документы, содержащие рекомендации или предложения; к ним относятся методические документы;
- документы, содержащие информацию о том, как последовательно выполнять действия и процессы; такие документы могут включать документированные процедуры, рабочие инструкции и чертежи;
- документы, содержащие объективные свидетельства выполненных действий или достигнутых результатов; к таким документам относятся записи.

Каждая организация определяет объем необходимой документации и ее носители. Это зависит от таких факторов, как вид и размер организации, сложность и взаимодействие процессов, сложность продукции, требования потребителей, соответствующие обязательные требования, продемонстрированные способности персонала, а также от глубины, до которой необходимо подтверждать выполнение требований к системе менеджмента качества.

Оценивание систем менеджмента качества. При оценке систем менеджмента качества следует задавать четыре основных вопроса в отношении каждого оцениваемого процесса:

1. Выявлен и определен ли соответствующим образом процесс?
2. Распределена ли ответственность?
3. Внедрены и поддерживаются ли в рабочем состоянии процедуры?
4. Эффективен ли процесс в достижении требуемых результатов?

Совокупные ответы на приведенные выше вопросы могут определить результаты оценивания. Оценка системы менеджмента качества может различаться по области применения и включать такие виды деятельности, как аудит (проверку) и анализ системы менеджмента качества, а также самооценку.

Аудиты (проверки) применяют для определения степени выполнения требований к системе менеджмента качества. Наблюдения аудитов (проверок) используют для оценки эффективности системы менеджмента качества и определения возможностей для улучшения.

Аудиты (проверки), проводимые первой стороной (самой организацией) или от ее имени для внутренних целей, могут служить основой для декларирования организацией о своем

соответствии.

Аудиты (проверки), проводимые второй стороной, могут проводиться потребителями организации или другими лицами от имени потребителей.

Аудиты (проверки), проводимые третьей стороной, осуществляются внешними независимыми организациями. Такие организации, обычно имеющие аккредитацию, проводят сертификацию на соответствие требованиям, например требованиям ГОСТ Р ИСО 9001.

ИСО 19011 содержит методические указания по аудиту (проверке).

Одна из задач высшего руководства – проведение регулярного систематического оценивания пригодности, адекватности, эффективности и результативности системы менеджмента качества с учетом политики и целей в области качества. Этот анализ может включать рассмотрение необходимости адаптации политики и целей в области качества в ответ на изменение потребностей и ожиданий заинтересованных сторон. Анализ включает определение потребности в действиях.

При анализе системы менеджмента качества наряду с другими источниками информации используют отчеты по аудитам (проверкам).

Самооценка организации является всесторонним и систематическим анализом деятельности организации и результатов по отношению к системе менеджмента качества или модели совершенства (модели премии по качеству).

Самооценка может дать общее представление о деятельности организации и степени развития системы менеджмента качества. Она может также помочь определить организации области, нуждающиеся в улучшении, и приоритеты.

Постоянное улучшение. Целью постоянного улучшения системы менеджмента качества является увеличение возможности повышения удовлетворенности потребителей и других заинтересованных сторон. Действия по улучшению включают:

- анализ и оценку существующего положения для определений областей для улучшения;
- установление целей улучшения;
- поиск возможных решений для достижения целей;
- оценивание и выбор решений;
- выполнение выбранных решений;
- измерение, проверку, анализ и оценку результатов выполнения для установления того, что достигнуты ли цели;
- оформление изменений.

Результаты анализируют с целью установления дальнейших возможностей для улучшения. Таким образом, улучшение является постоянным действием. Аудиты (проверки) и анализ системы менеджмента качества могут также использоваться для определения возможностей улучшения.

Роль статистических методов. Использование статистических методов может помочь в понимании изменчивости и, следовательно, может помочь организациям в решении проблем и повышении результативности и эффективности. Эти методы также способствуют лучшему применению имеющихся в наличии данных для оказания помощи в принятии решений.

Изменчивость можно наблюдать в ходе и результатах многих видов деятельности, даже в условиях очевидной стабильности. Такую изменчивость можно проследить в измеряемых характеристиках продукции и процессов. Ее наличие можно заметить на различных стадиях жизненного цикла продукции от исследования рынка до обслуживания потребителей и утилизации.

Статистические методы могут помочь при измерении, описании, анализе, интерпретации и моделировании такой изменчивости даже при относительно ограниченном количестве данных. Статистический анализ таких данных может помочь лучше понять природу, масштаб и причины изменчивости, способствуя таким образом решению, и даже предупреждению проблем, которые могут быть результатом такой изменчивости, а также постоянному улучшению.

Методические указания по применению статистических методов в системе менеджмента качества приведены в ИСО/ТО 10017.

Направленность систем менеджмента качества и других систем менеджмента. Система менеджмента качества является частью системы менеджмента организации, которая

направлена на достижение результатов в соответствии с целями в области качества, чтобы удовлетворять потребности, ожидания и требования заинтересованных сторон. Цели в области качества дополняют другие цели организации, связанные с развитием, финансированием, рентабельностью, окружающей средой, охраной труда и безопасностью. Различные части системы менеджмента организации могут быть интегрированы вместе с системой менеджмента качества в единую систему менеджмента, использующую общие элементы. Это может облегчить планирование, выделение ресурсов, определение дополнительных целей и оценку общей эффективности организации. Система менеджмента организации может быть оценена на соответствие собственным требованиям организации. Она может быть также проверена на соответствие требованиям ГОСТ Р ИСО 9001 и ГОСТ Р ИСО 14001. Эти аудиты (проверки) могут проводиться отдельно или совместно.

Взаимосвязь между системами менеджмента качества и моделями совершенства. Подходы систем менеджмента качества, приведенные в семействе стандартов ИСО 9000, и модели совершенства основаны на общих принципах. Оба эти подхода:

- дают возможность организации выявить свои сильные и слабые стороны;
- содержат положения по оцениванию в сравнении с общими моделями;
- обеспечивают основу для постоянного улучшения;
- включают способы внешнего признания.

Различие между подходами систем менеджмента качества семейства ИСО 9000 и моделями совершенства заключается в их областях применения.

Стандарты семейства ИСО 9000 содержат требования к системам менеджмента качества и рекомендации по улучшению деятельности; оценивание систем менеджмента качества устанавливает выполнение этих требований. Модели совершенства содержат критерии, позволяющие проводить сравнительную оценку деятельности организации, и это применимо ко всем видам деятельности и ко всем заинтересованным сторонам. Критерии оценки в моделях совершенства обеспечивают организации основу для сравнения ее деятельности с деятельностью других организаций.

1.3.2 ГОСТ Р ИСО/МЭК ТО 15504

Основы оценки и аттестации зрелости процессов создания и сопровождения программных средств и информационных систем установлены стандартами ИСО/МЭК ТО 15504, разработанными на базе концепций CMM (Capability Maturity Model for Software).

1.3.2.1 Обзор

ИСО/МЭК ТО 15504 предоставляет основу для аттестации процесса жизненного цикла программных средств. Эта основа может быть использована организациями, занимающимися планированием, управлением, наблюдением, контролем и совершенствованием приобретения, поставки, разработки, эксплуатации, развития и поддержки программных средств.

ИСО/МЭК ТО 15504 предоставляет структурный подход к аттестации процесса жизненного цикла программных средств, проводящейся:

- организацией или от ее имени с целью выяснения состояния ее собственных процессов для их усовершенствования;
- организацией или от ее имени с целью определения пригодности ее собственных процессов для выполнения определенного требования или класса требований;
- организацией или от ее имени с целью определения пригодности процессов другой организации для определенного договора или класса договоров.

Такая основа для аттестации процесса способствует самоаттестации; направлена на обеспечение адекватности управления аттестуемыми процессами; принимает во внимание контекст, в котором выполняются аттестуемые процессы; выдает набор рейтингов процесса (профиль процесса), а не результат типа зачет/незачет; подходит ко всем сферам приложений и для организаций любого размера.

Чтобы организация могла улучшить качество своей продукции, она должна иметь проверенный, последовательный и надежный метод для аттестации состояния своих процессов, а также она должна иметь средства использования результатов как часть связной программы

усовершенствования.

Использование аттестации процессов внутри организации должно способствовать:

- выработке культуры постоянного совершенствования, а также соответствующих механизмов поддержания этой культуры;
- разработке процессов, отвечающих бизнес-целям;
- оптимизации использования ресурсов.

Это приведет к появлению зрелых организаций, максимально восприимчивых к требованиям потребителя и рынка, имеющих минимальную стоимость полного жизненного цикла своей продукции и, как результат, максимально удовлетворяющих конечного пользователя.

Покупателям также будет выгодно использовать аттестацию процесса. Ее применение при определении зрелости

- уменьшит неопределенность при выборе поставщиков программнонасыщенных систем за счет того, что риски, связанные со зрелостью подрядчика, выявляются еще до заключения договора;
- позволит заранее предусмотреть необходимые меры на случай возникновения рискового события;
- предоставит количественные критерии выбора при сопоставлении потребностей бизнеса, требований и оценочной стоимости проекта со зрелостью конкурирующих поставщиков.

Главные преимущества стандартизированного подхода к аттестации процесса в том, что он:

- станет открытым, общедоступным подходом к аттестации процесса;
- приведет к общему пониманию использования аттестации для усовершенствования процесса и оценки зрелости;
- при приобретении облегчит оценку зрелости поставщика;
- будет контролироваться и регулярно пересматриваться в свете опыта использования;
- будет меняться только международным соглашением;
- будет способствовать согласованию существующих схем.

Подход к аттестации процесса, определенный в ИСО/МЭК ТО 15504, разработан с целью предоставить основу для общего подхода к описанию результатов аттестации процесса, позволяющего в какой-то степени сравнивать аттестации, основанные на разных, но совместимых моделях и методах. Изощренность и сложность, требующиеся от процесса, зависят от его контекста. Например, планирование для группы из пяти человек необходимо гораздо в меньшем объеме, чем для группы из пятидесяти человек. Этот контекст влияет на то, как ведущий аттестатор судит о действии, а также на степень сравнимости профилей различных процессов.

1.3.2.2 Область применения

Аттестация процесса применяется в основном в двух контекстах (рисунок 1.17).

Рисунок 1.17 - Оценка и аттестация процессов жизненного цикла программных средств и информационных систем

В контексте усовершенствования процесса аттестация процесса предоставляет средства охарактеризовать текущую деятельность организационной единицы в терминах зрелости некоторых выбранных процессов. Анализ результатов в свете бизнеспотребностей организации выявляет сильные и слабые стороны процессов, а также присущие им риски. Это, в свою очередь, позволяет определить, эффективны ли эти процессы в достижении своих целей, а также выявить существенные причины низкого качества или превышения бюджета или сроков. Все вместе это позволяет расставить приоритеты при усовершенствовании процессов.

Определение зрелости процесса связано с анализом соответствия заявленной зрелости выбранных процессов целевому профилю зрелости процессов для того, чтобы выявить риски, связанные с выполнением проекта, использующего выбранные процессы. Заявленная зрелость может быть основана на результатах прежних аналогичных аттестаций процесса или на аттестации, проведенной с целью выработки заявленной зрелости.

Две части ИСО/МЭК ТО 15504 (части 7 и 8) посвящены использованию аттестации для усовершенствования процесса и для определения зрелости процесса. Другие части ИСО/МЭК ТО 15504 посвящены различным вопросам, относящимся к аттестации процесса.

ИСО/МЭК ТО 15504 разработан так, чтобы в рамках единого источника удовлетворить потребности потребителей, поставщиков и аттестаторов, а также их индивидуальные требования. Преимущества, вытекающие из использования данного комплекта документов, включают:

Для приобретателей:

- способность определять текущую и потенциальную зрелости процессов жизненного цикла у поставщика;

Для поставщиков:

- способность определять текущую и потенциальную зрелости своих собственных процессов жизненного цикла;
- способность определять области и приоритеты для усовершенствования процессов;
- основу, задающую схему усовершенствования процессов;

Для аттестаторов:

- основу для проведения аттестаций.

1.3.2.3 Состав ИСО/МЭК ТО 15504

ИСО/МЭК ТО 15504 состоит из девяти частей. В данном разделе описана каждая часть и

ее роль в ИСО/МЭК ТО 15504.

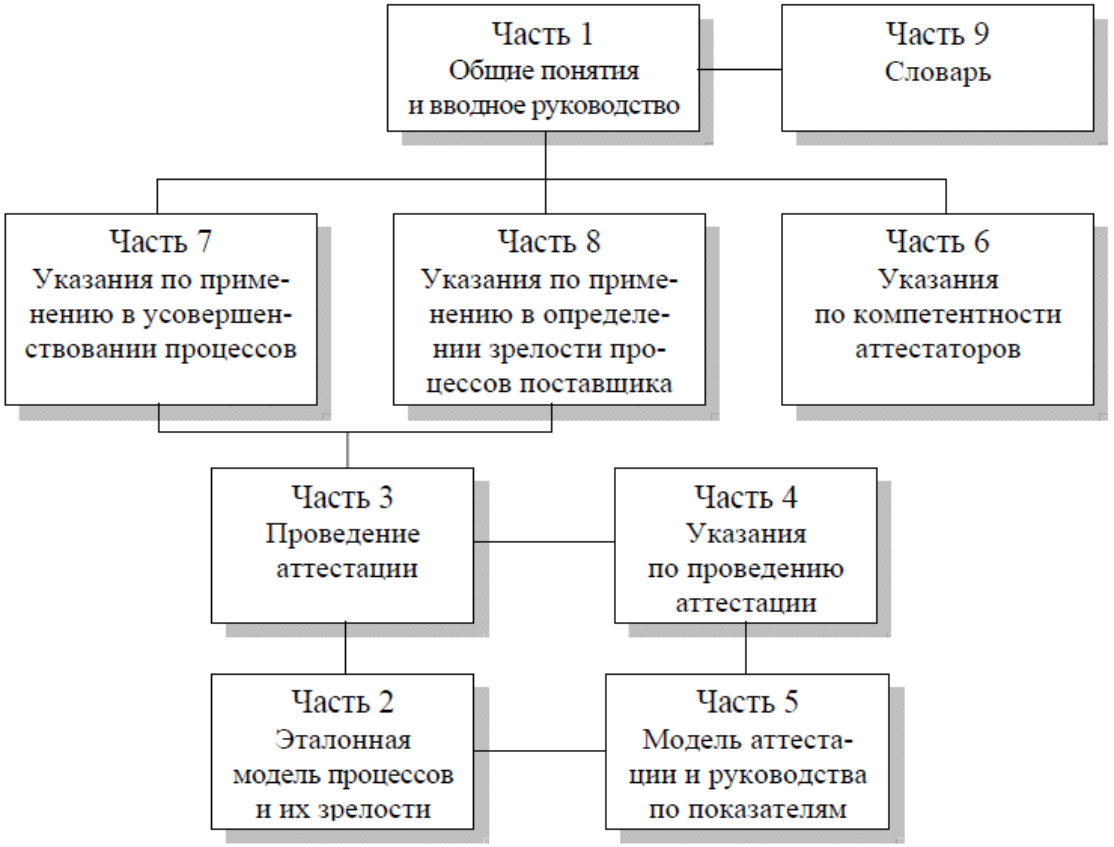


Рисунок 1.18 - Состав ИСО/МЭК ТО 15504

Рисунок 1.18 показывает потенциальную схему использования ИСО/МЭК ТО 15504. Часть 1 (этот документ) служит общей отправной точкой ИСО/МЭК ТО 15504. Читатели, интересующиеся только усовершенствованием процесса или определением зрелости поставщика, должны прочесть части 7 и 8 соответственно, содержащие детальное руководство по этим контекстам использования. Эти части позволят пользователю определить наиболее подходящий для себя способ использования нормативных компонентов ИСО/МЭК ТО 15504 (части 2 и 3). Часть 4 является руководством по применению части 2 и части 3, тогда как часть 5 – это пример аттестационной модели, совместимой с эталонной моделью (часть 2). Пользователи, интересующиеся в основном ролью аттестаторов, должны обратить внимание на часть 6, содержащую руководство по навыкам и компетентности аттестаторов.

В таблице 1.2 перечислены основные классы читателей ИСО/МЭК ТО 15504 и указаны, какие части данного набора документов посвящены первостепенным областям их интересов.

Таблица 1.2 Аудитория ИСО/МЭК ТО 15504

Класс читателей	Интересы	Части, предлагаемые к прочтению

Заказчик (спонсор) аттестации	Как проводится аттестация, какие требуются инструменты и иная поддержка, как инициировать аттестацию	1, 2, 3, 4, 6
Заказчик (спонсор) усовершенствования процесса	Инициирование программы усовершенствования, определение входов для аттестации для целей усовершенствования, использование результатов аттестации для усовершенствования	7
Заказчик (спонсор) определения зрелости процесса	Инициирование программы определения зрелости поставщика, определение целевого профиля зрелости, проверка и использование результатов аттестации при определении зрелости	
Аттестаторы	Проведение согласованной аттестации, развитие навыков и компетентности, необходимых для проведения аттестации	8
	Разработка модели для проведения аттестации, совместимой с эталонной моделью	
Разработчики аттестационных моделей	Разработка метода, который поддерживал бы проведение аттестации, соответствующей требованиям	2, 3, 4, 5, 6
Разработчики методов аттестации	Разработка инструментальных средств, которые помогали бы аттестаторам в сборе, записи, классификации данных в ходе аттестации	2, 3, 4, 5
Разработчики инструментальных средств		2, 3, 4
		2, 3, 4, 5

Часть 1 (информационная) является отправной точкой ИСО/МЭК ТО 15504. Она описывает взаимодействие частей набора документов и содержит руководство по их выбору и использованию. Она разъясняет требования, содержащиеся в ИСО/МЭК ТО 15504 и их применимость к проведению аттестаций.

Часть 2 (нормативная) ИСО/МЭК ТО 15504 определяет двумерную эталонную модель для описания процессов и их зрелости, использующуюся при аттестации процессов. Эталонная модель определяет ряд процессов, установленных в терминах их назначения и итогов, а также основу для оценивания зрелости процессов посредством аттестации атрибутов процессов, структурированных по уровням зрелости. Также обозначены требования для установления совместимости различных аттестационных моделей с эталонной моделью.

Часть 3 (нормативная) ИСО/МЭК ТО 15504 определяет требования для проведения аттестации таким образом, чтобы результаты были повторимыми, надежными и согласующимися.

Часть 4 (информационная) ИСО/МЭК ТО 15504 содержит руководство по проведению аттестаций процессов жизненного цикла программных средств, по интерпретации требований ИСО/МЭК ТО 15504-2 и ИСО/МЭК ТО 15504-3 для различных контекстов аттестации. Это руководство охватывает выбор и использование документированного процесса для аттестации, совместимой аттестационной модели (или моделей), а также вспомогательного инструмента или средства аттестации. Это руководство является достаточно общим и применимо для всех организаций для проведения аттестаций с использованием

разнообразных методов и технических приемов для того, чтобы поддерживаться целым рядом средств.

Часть 5 (информационная) ИСО/МЭК ТО 15504 содержит пример модели для проведения аттестации процесса, основанную на эталонной модели из ИСО/МЭК ТО 15504-2 и непосредственно с ней совместимую. Аттестационная модель (или модели) расширяет эталонную модель включением в нее всеобъемлющего набора показателей производительности и зрелости процессов.

Часть 6 (информационная) ИСО/МЭК ТО 15504 описывает компетентность, образование, специальную подготовку и опыт, необходимые аттестаторам для проведения аттестации процессов. В ней приведены механизмы, которые могут быть использованы для демонстрации компетентности и подтверждения образования, специальной подготовки и опыта.

Часть 7 (информационная) ИСО/МЭК ТО 15504 описывает, как определять входы и использовать результаты аттестации, имеющей целью усовершенствование процесса. Это руководство включает примеры применения усовершенствования процесса в различных ситуациях.

Часть 8 (информационная) ИСО/МЭК ТО 15504 описывает, как определять входы и использовать результаты аттестации, имеющей целью определение зрелости процессов. Она охватывает определение зрелости процессов как в простейших ситуациях, так и в более сложных, включающих, например, будущую зрелость. Это руководство по проведению определения зрелости процессов может использоваться либо организацией для определения собственной зрелости, либо потребителем для определения зрелости (потенциального) поставщика.

Часть 9 (нормативная) ИСО/МЭК ТО 15504 является консолидированным словарем терминов, специфически определенных для целей ИСО/МЭК ТО 15504.

1.3.2.4 Связь с другими международными стандартами

ИСО/МЭК ТО 15504 дополняет некоторые международные стандарты и другие модели для оценки зрелости и эффективности организаций и процессов. Данный раздел описывает связь между ИСО/МЭК ТО 15504 и большинством связанных с ним международных стандартов.

ИСО/МЭК ТО 15504 преследует ту же цель, что и серия стандартов ИСО 9000 – обеспечить уверенность в системе управления качеством поставщика, одновременно предоставляя потребителям основу для оценки того, обладают ли потенциальные поставщики производственными возможностями, отвечающими их потребностям. Аттестация процессов дает пользователям возможность оценивать зрелость процессов по непрерывной шкале таким образом, что эти оценки сравнимы и повторимы в отличие от аудитов качества, основанных на ИСО 9001:1994, дающих оценку типа зачет/незачет. Кроме того, основа, описываемая ИСО/МЭК ТО 15504, предоставляет возможность подобрать объем аттестации так, чтобы он охватывал лишь определенные процессы, вызывающие интерес, а не все процессы, используемые организационной единицей.

ИСО/МЭК ТО 15504 в части «Процессного подхода» и «Системного подхода к административному управлению» отвечает концепции ИСО 2000 года в области системы качества. Требования к системам административного управления (менеджмента) качеством установлены в стандартах:

ISO 9000:2000 Quality management systems – Fundamentals and vocabu-lary (Системы административного управления качеством. Основы и словарь);

ISO 9000:2000 Quality management systems – Requirements (Системы административного управления качеством. Требования);

ISO 9000:2000 Quality management systems – Guidelines for performance improvement (Системы административного управления качеством. Руково- дящие указания по усовершенствованию);

ISO 19011:2000 Quality management systems – Guidelines for auditing quality and environmental management systems.

Эталонная модель описания, оценки и аттестации зрелости процессов деятельности, используемой при выполнении этапов жизненного цикла продукции, проекта или системы (ИСО/МЭК ТО 15504-2), согласована с ИСО/МЭК 12207:1995 «Информационная технология. Процессы жизненного цикла программных средств». Однако эта эталонная модель может быть

использована для разработки модели оценки уровня зрелости процессов других видов деятельности. Изложенные в ИСО/МЭК ТО 15504 рабочие продукты процессов и их характеристики могут быть использованы для оценки и усовершенствования процессов любого вида деятельности. ИСО/МЭК ТО 15504 предоставляет механизм включения в этот перечень дополнительных рабочих продуктов и процессов, необходимых для осуществления конкретного вида деятельности и, следовательно, для оценки уровня зрелости этих дополнительных процессов и вида деятельности в целом.

Концептуальные основы системы административного управления качеством определены в восьми принципах. ИСО определяет принцип административного управления качеством как «всеобъемлющее и фундаментальное правило или убеждение, применяемое при руководстве и управлении организацией, направленное на непрерывное и долгосрочное улучшение ее производительности путем ориентации на потребителей одновременно с удовлетворением потребностей остальных участвующих сторон». Эти принципы административного управления качеством войдут в документ ISO 9000 2000 года. На этих принципах будет базироваться семейство стандартов ISO 9000.

Выполнение этих принципов будет способствовать повышению управленческой культуры, проникновению системы административного управления качеством во все виды деятельности организации (т.е. выполнения концепции всеобщего административного управления качеством, TQM – Total Quality Management) и, как следствие, обеспечению конкурентоспособности создаваемой организацией продукции, проектов, систем и услуг.

Принцип 1. Ориентация организации на потребителя (Customer-Focused Organization)

«Организации зависят от своих потребителей и, таким образом, должны понимать текущие и будущие потребности потребителей, удовлетворять их требования и стремиться превзойти их ожидания». Применение принципа ориентации организации на потребителя приводит к следующим действиям:

- понимание всего спектра потребностей и ожиданий потребителей относительно продуктов, поставок, цен, надежности и т. д.;
- обеспечение взвешенного подхода к потребностям и ожиданиям потребителей и других участвующих сторон (владельцев, персонала, поставщиков, местных сообществ и общества в целом);
- распространение информации об этих потребностях и ожиданиях во всей организации;
- измерение степени удовлетворенности потребителей и влияние на результат;
- управление взаимоотношениями с потребителями.

Полезные применения данного принципа включают:

- обеспечение того, что потребности потребителей и других участвующих сторон осознаются всей организацией, для формулировки политики и стратегии (policy and strategy formulation);
- обеспечение непосредственной связи соответствующих целей и плановых показателей с потребностями и ожиданиями потребителей, для установления целей и плановых показателей (goal and target setting);
- повышение производительности организации для удовлетворения потребностей потребителей, для управления операциями (operational management);
- обеспечение того, что персонал обладает знаниями и опытом, требующимися для удовлетворения потребителей организации, для управления людскими ресурсами (human resource management).

Принцип 2. Лидерство (Leadership)

«Лидеры организаций обеспечивают единство назначения и направления организации. Они должны создать и поддерживать внутреннюю окружающую среду, в которой люди могут в полной мере участвовать в достижении стратегических целей организации».

Применение принципа лидерства приводит к следующим действиям:

- действенность и личный пример;
- понимание изменений во внешней окружающей среде и реагирование на них;
- внимание к потребностям всех участвующих сторон, включая потребителей, владельцев, персонал, поставщиков, местные сообщества и общество в целом;
- выработка ясного видения будущего организации;
- выработка общих ценностей и этики на всех уровнях организации;

- установление доверия и искоренение страха;
- обеспечение персонала требуемыми ресурсами и свободой, необходимыми для того, чтобы действовать ответственно и обоснованно;
- воодушевление, поощрение и признание вклада персонала;
- содействие открытому и честному общению;
- обучение, подготовка и инструктирование персонала;
- выработка достойных целей и плановых показателей;
- реализация стратегии по достижению этих целей и плановых показателей.

Полезные применения данного принципа включают:
- выработку и распространение ясного видения будущего организации, для формулировки политики и стратегии;
- преобразование видения организации в измеримые цели и плановые показатели, для установления целей и плановых показателей;
- стратегические цели организации достигаются полномочным и вовлеченным персоналом, для управления операциями;
- наличие полномочной, мотивированной, хорошо информированной и стабильной рабочей силы, для управления людскими ресурсами.

Принцип 3. Вовлечение персонала (Involvement of People)

«Люди составляют сущность организации на всех уровнях, и их полная вовлеченность способствует применению их способностей на благо организации».

Применение принципа вовлечения персонала приводит к следующим действиям персонала:
- принятие на себя задач и ответственность за их решение;
- активный поиск возможностей усовершенствования;
- активный поиск возможностей повышения собственных квалификации, знаний и опыта;
- свободный обмен знаниями и опытом в группах и коллективах;
- концентрация на создании ценностей для потребителя;
- новаторство и творчество в дальнейшем продвижении стратегических целей организации;
- олицетворение организации перед лицом потребителей, местных сообществ и общества в целом;
- получение удовольствия от своей работы;
- гордость и удовлетворение быть частью организации.

Полезные применения данного принципа включают:
- персонал, эффективно участвующий в усовершенствовании политики и стратегии организации, для формулировки политики и стратегии;
- персонал, принимающий на себя задачи и разделяющий ответственность за их решение, для установления целей и плановых показателей;
- персонал, вовлеченный в соответствующие усовершенствования решений и процессов, для управления операциями;
- персонал, в большей степени удовлетворенный своей работой и активно вовлеченный в собственный рост и развитие на благо организации, для управления людскими ресурсами.

Принцип 4. Процессный подход (Process Approach)

«Желаемый результат достигается более эффективно, когда связанные ресурсы и деятельность управляются как процесс».

Применение принципа процессного подхода приводит к следующим действиям персонала:
- определение процесса достижения желаемого результата;
- выявление и измерение входов и выходов процесса;
- выявление интерфейсов процесса с функциями организации;
- оценка возможного риска, его последствий и влияния процесса на потребителей, поставщиков и другие участвующие в процессе стороны;
- четкое распределение ответственности, полномочий и подотчетности при управлении процессом;
- выявление внутренних и внешних потребителей, поставщиков и других участвующих в процессе сторон;
- при проектировании процессов уделяется внимание шагам процессов, видам

деятельности, потокам, контрольным величинам, потребностям в подготовке персонала, оборудования, методах, информации, материалах и других ресурсах, необходимых для достижения желаемого результата.

Полезные применения данного принципа включают:

- использование определенных процессов во всей организации приведет к более предсказуемым результатам, лучшему использованию ресурсов, сокращению времени цикла и снижению затрат, для формулировки политики и стратегии;
- понимание зрелости процессов способствует выработке достойных целей и плановых показателей, для установления целей и плановых показателей;
- принятие процессного подхода ко всем операциям приводит к снижению затрат, предотвращению ошибок, контролю за отклонениями, сокращению времени цикла и более предсказуемым выходам, для управления операциями;
- установление эффективных по затратам процессов для управления людскими ресурсами, таких как найм, образование и подготовка, способствует приведению этих процессов в соответствие потребностям организации и создает более зрелую рабочую силу, для управления людскими ресурсами.

Принцип 5. Системный подход к административному управлению (System Approach to Management)

«Выявление, понимание и административное управление системой взаимосвязанных процессов для заданной стратегической цели повышает эффективность и результативность организации».

Применение принципа системного подхода к административному управлению приводит к следующим действиям:

- определение системы путем выявления или разработки процессов, влияющих на достижение заданной стратегической цели;
- структурирование системы так, чтобы достичь заданную стратегическую цель наиболее эффективным способом;
- понимание взаимозависимостей между процессами системы;
- непрерывное усовершенствование системы посредством измерения и оценки;
- предварительное установление ограничений по ресурсам.

Полезные применения данного принципа включают:

- создание всеобъемлющих и достойных планов, связывающих входы функций и процессов, для формулировки политики и стратегии;
- цели и плановые показатели конкретных процессов приведены в соответствие с ключевыми стратегическими целями организации, для установления целей и плановых показателей;
- более широкий взгляд на эффективность процессов, что приводит к пониманию причин проблем и своевременным действиям по усовершенствованию, для управления операциями;
- лучшее понимание распределения ролей и ответственности при достижении общих стратегических целей, уменьшая таким образом межфункциональные барьеры и улучшая коллективную работу, для управления людскими ресурсами.

Принцип 6. Непрерывное усовершенствование (Continual Improvement)

«Непрерывное усовершенствование должно быть постоянной стратегической целью организации».

Применение принципа непрерывного усовершенствования приводит к следующим действиям:

- превращение непрерывного усовершенствования продуктов, процессов и систем в стратегическую цель каждого сотрудника организации;
- применение базовых понятий последовательного (инкрементного) и скачкообразного усовершенствования;
- проведение периодических аттестаций степени достижения установленных критериев высшего качества для выявления областей потенциального усовершенствования;
- непрерывное повышение эффективности и результативности всех процессов;
- поощрение действий, основанных на предотвращении проблем;
- предоставление каждому члену организации необходимых образования и подготовки по методам и инструментальным средствам непрерывного усовершенствования, таким, как: цикл «планирование – исполнение – проверка – корректирующие действия» (Plan – Do – Check – Act);

- решение проблем;
- реинжиниринг процессов;
- нововведение в процессах;
- установление мер и целей для направления и отслеживания усовершенствований;
- признание усовершенствований.

Полезные применения данного принципа включают:

- создание и осуществление более конкурентоспособных бизнес-планов путем объединения непрерывного усовершенствования со стратегическим и бизнес-планированием, для формулировки политики и стратегии;
- установление реалистичных и достойных целей усовершенствования и предоставление ресурсов для их достижения, для установления целей и плановых показателей;
- вовлечение персонала организации в непрерывное усовершенствование процессов, для управления операциями;
- обеспечение всего персонала организации инструментальными средствами, возможностями и мотивацией для совершенствования продуктов, процессов и систем, для управления людскими ресурсами.

Принцип 7. Основанный на фактах подход к принятию решений (Factual Approach to Decision Making)

«Эффективные решения базируются на анализе данных и информации».

Применение данного принципа приводит к следующим действиям:

- осуществление измерений и сбор данных и информации, относящихся к стратегической цели;
- обеспечение существенной точности, надежности и доступности данных и информации;
- анализ данных и информации с применением обоснованных методов;
- понимание значения соответствующих статистических методик;
- принятие решений и осуществление действий на базе логического анализа, уравновешенного опытом и интуицией.

Полезные применения данного принципа включают:

- стратегии, основанные на соответствующих данных и информации более реалистичны и достижимы с большей вероятностью, для формулировки политики и стратегии;
- применение соответствующих сравнительных данных и информации для установления реалистичных и достойных целей и плановых показателей, для установления целей и плановых показателей;
- данные и информация являются базисом для понимания производительности процессов и систем для направления усовершенствований и предотвращения будущих проблем, для управления операциями;
- анализ данных и информации из таких источников, как опросы персонала, предложения сотрудников, целевых групп для направления формулировки политики управления людскими ресурсами, для управления людскими ресурсами.

Принцип 8. Взаимовыгодные отношения с поставщиками (Mutually Beneficial Supplier Relationship)

«Организация и ее поставщики взаимозависимы, и взаимовыгодные отношения повышают способность обоих производить ценности».

Применение принципа взаимовыгодных отношений с поставщиками

приводит к следующим действиям:

- выявление и выбор ключевых поставщиков;
-установление с поставщиками отношений, которые бы уравновешивали краткосрочные выгоды и долгосрочные соображения для организации и общества в целом;
- создание ясного и открытого общения;
- инициация совместных разработок и усовершенствования продуктов и процессов;
- совместное установление ясного понимания потребностей потребителей;
- обмен информацией и будующими планами;
- признание усовершенствований и достижений поставщиков.

Полезные применения данного принципа включают:

- здание конкурентоспособных преимуществ путем организации стратегических союзов или партнерских отношений с поставщиками, для формулировки политики и стратегии;

- установление более достойных целей и плановых показателей с помощью вовлечения и участия поставщиков на ранних этапах, для установления целей и плановых показателей;
- создание отношений с поставщиками и управление этими отношениями для обеспечения надежных, своевременных и свободных от дефектов поставок, для управления операциями;
- выработку и повышение зрелости поставщиков посредством проведения подготовки поставщиков и совместных усилий по усовершенствованию, для управления людскими ресурсами.

Для комплексного обеспечения нормативной базы практической реализации концепции ИСО/МЭК 15504 с учетом принципов системы административного управления качеством необходимо предусмотреть также выполнение требований нижеприведенных международных стандартов:

ISO/TR 10006:1997 Quality management – Guidelines to quality in project management (Административное управление качеством. Руководящие указания по качеству при административном управлении проектами);

ISO 10007:1997 Quality management – Guidelines for configuration management (Административное управление качеством. Руководящие указания административного управления конфигурацией);

ISO/IEC TR 16326:1999 Software engineering – Guide for the application of ISO/IEC 12207 to project management (Программная инженерия. Руководство по приложению ИСО/МЭК 12207 к административному управлению проектами);

ISO/IEC TR 15271:1998 Information technology – Guide for ISO/IEC

12207 (Software Life Cycle Processes) (Информационная технология. Руководство по применению ИСО/МЭК 12207);

ISO/IEC TR 15846:1998 Information technology – Software life cycle processes – Configuration Management (Информационная технология. Процессы жизненного цикла программных средств. Управление конфигурацией); ISO/IEC 12119:1994 Information technology – Software packages – Quality requirements and testing (Информационная технология. Пакеты программных средств. Требование к качеству и тестирование);

ISO/IEC TR 14759:1999 Software engineering – Mock up and prototype – A categorization of software mock up and prototype models and their use (Программная инженерия. Макетирование и прототипирование);

ISO/IEC 9126:1991 Information technology – Software product evaluation – Quality characteristics and guidelines for their use (Информационная технология. Оценка программного продукта. Характеристики качества и руководящие указания по их применению);

ISO/IEC 14598:1999 Information technology – Software product evaluation

– Part 1–5 (Информационная технология. Оценка программной продукции); ISO/IEC 15026:1998 Information technology – System and software integrity levels (Информационная технология. Уровни целостности программных средств и систем);

ISO/IEC 14764:1999 Information technology – Software maintenance (Информационная технология. Сопровождение программных средств);

ISO/IEC TR 14471:1999 Information technology – Software engineering – Guidelines for the adoption of CASE tools (Информационная технология. Программная инженерия. Руководящие указания по адаптации инструментальных средств CASE);

ISO/IEC TR 9294:1990 Information technology – Guidelines for the management of software documentation (Информационная технология. Руководящие указания по административному управлению программной документацией);

ISO/IEC 6592:2000 Information technology – Guidelines for the documentation of computer-based application systems (Информационная технология. Руководящие указания по документированию прикладных информационных систем);

ISO/IEC 15910:1999 Information technology – Software user documentation process (Информационная технология. Пользовательская документация программных средств);

ISO/IEC 14756:1999 Information technology – Measurement and rating of performance of computer-based software systems (Информационная технология.

Измерение и определение рейтинга программных средств информационных систем).

Необходимо рассмотреть возможность применения ИСО/МЭК ТО

15504 в индустрии различных областей промышленности, отвечающих требованиям CALS-технологии (стандарты серии 10303).

Особое внимание необходимо обратить на создание единого и гармонизированного понятийно-терминологического аппарата в области индустрии программной продукции и информационных технологий.

1.3.3 ГОСТ Р ИСО/МЭК 12207–99. Информационная технология. Процессы жизненного цикла программных средств

Разработан Всероссийским научно-исследовательским институтом стандартизации (ВНИИстандарт) Госстандарта России. Настоящий стандарт содержит полный аутентичный текст международного стандарта ИСО/МЭК 12207–95 «Информационная технология. Процессы жизненного цикла программных средств».

1.3.3.1 Введение

Программные средства являются неотъемлемыми частями информационных технологий и традиционных систем, таких как транспортные, военные, финансовые и система здравоохранения. При этом подразумевается усиление роли стандартов, процедур, методов, средств (инструментария) и внешних условий для разработки и сопровождения программных средств (программного обеспечения). Подобная многоплановость подходов создает значительные трудности при управлении программными средствами и в технологиях программирования, особенно при интеграции продуктов и услуг. Требуется определенное упорядочение вопросов создания программных средств при переходе от подобной многоплановости к общей структуре, которая может быть использована профессионалами для «разговора на одном языке» при создании и управлении программными средствами. Настоящий стандарт устанавливает такую общую структуру.

Данная структура охватывает жизненный цикл программных средств от концепции замыслов через определение и объединение процессов до заказа и поставки программных продуктов и услуг. Кроме того, данная структура предназначена для контроля и модернизации данных процессов.

Процессы, определенные в настоящем стандарте, образуют множество общего назначения. Конкретная организация, в зависимости от своих целей, может выбрать соответствующее подмножество процессов для выполнения своих конкретных задач. Поэтому настоящий стандарт следует адаптировать для конкретной организации, проекта или приложения. Настоящий стандарт предназначен для использования как в случае отдельно поставляемых программных средств, так и для программных средств, встраиваемых или интегрируемых в общую систему.

1.3.3.2 Область применения

Назначение. Настоящий стандарт устанавливает, используя четко определенную терминологию, общую структуру процессов жизненного цикла программных средств, на которую можно ориентироваться в программной индустрии. Настоящий стандарт определяет процессы, работы и задачи, которые используются: при приобретении системы, содержащей программные средства, или отдельно поставляемого программного продукта; при оказании программной услуги, а также при поставке, разработке, эксплуатации и со- провождении программных продуктов. Понятие программных средств также охватывает программный компонент программно-аппаратных средств.

Настоящий стандарт также определяет процесс, который может быть использован при определении, контроле и модернизации процессов жизненного цикла программных средств.

Область распространения. Настоящий стандарт применяется при приобретении систем, программных продуктов и оказании соответствующих услуг, а также при поставке, разработке, эксплуатации и сопровождении программных продуктов и программных компонентов программно-аппаратных средств как в самой организации, так и вне ее. Стандарт содержит также те аспекты описания системы, которые необходимы для обеспечения понимания сути программных продуктов и услуг.

Стандарт также применяется при двусторонних отношениях и может в равной степени применяться, если обе стороны принадлежат к одной и той же

организации. Диапазон применения может простираться от неформального соглашения о сотрудничестве до официально заключаемого контракта (договора). Стандарт может использоваться одной из сторон для самоконтроля.

Стандарт не распространяется на готовые программные продукты, если они не входят в поставляемый продукт.

Стандарт предназначен: для заказчиков систем, программных продуктов и услуг; поставщиков; разработчиков; операторов; персонала сопровождения; администраторов проектов; администраторов, отвечающих за качество, и пользователей программных продуктов.

Адаптация настоящего стандарта. Настоящий стандарт определяет набор процессов, работ и задач, предназначенных для адаптации к условиям конкретных программных проектов. Процесс адаптации заключается в исключении неприменяемых в условиях конкретного проекта процессов, работ и задач.

Соответствие. Соответствие настоящему стандарту определяется как выполнение всех процессов, работ и задач, выбранных из настоящего стандарта в процессе адаптации, для конкретного программного проекта. Осуществление процесса или работы считается завершенным, когда выполнены всетребуемые для них задачи в соответствии с предварительно установленными в договоре критериями и требованиями.

Любая организация (например, национальная, промышленная ассоциация, компания), применяющая настоящий стандарт в качестве условия обеспечения торговых сделок, обязана определить и опубликовать минимальный набор требуемых процессов, работ и задач, который обеспечивает проверку соответствия поставщика настоящему стандарту.

Ограничения. Настоящий стандарт описывает архитектуру процессов жизненного цикла программных средств, но не определяет детали реализации или выполнения работ и задач, входящих в данные процессы.

Стандарт не предназначен для определения наименований, форматов или подробного содержания выпускаемой документации. Стандарт может требовать разработки документов одного класса или типа, например различных планов, но не предусматривает, чтобы такие документы разрабатывались или комплектовались раздельно или совместно. Решение этих вопросов оставлено на усмотрение пользователей настоящего стандарта.

Стандарт не предопределяет конкретной модели жизненного цикла или метода разработки программного средства. Пользователи, применяющие настоящий стандарт, должны сами выбирать модель жизненного цикла применительно к своему программному проекту и распределять процессы, работы и задачи, выбранные из настоящего стандарта, на данной модели; выбирать и применять методы разработки программных средств и выполнять работы и задачи, соответствующие конкретному программному проекту.

Стандарт не имеет противоречий с существующими в организациях стратегиями, стандартами или процедурами. Однако любые возникающие конфликтные ситуации должны быть разрешены, а любые противоречащие условия и ситуации должны быть упомянуты в примечаниях как исключения при применении настоящего стандарта.

В тексте настоящего стандарта слово «должны» используется для выражения соглашения между двумя или более сторонами; слово «должна» – для выражения объявления цели или намерения одной из сторон; слово «следует» – для выражения рекомендации из имеющихся возможных вариантов; слово «может» – для обозначения образа действий, допускаемого в рамках ограничений настоящего стандарта.

В тексте настоящего стандарта приведен ряд перечней задач, однако ни один из перечней нельзя считать исчерпывающим, и они приведены в качестве примеров.

1.3.3.3 Прикладное применение настоящего стандарта

В настоящем разделе определяются процессы жизненного цикла программных средств, которые могут быть реализованы при заказе, поставке, разработке, эксплуатации и сопровождении программных продуктов. Целью настоящего раздела является создание «путеводителя» для пользователей, который поможет ориентироваться в стандарте и осмысленно применять его.

Построение стандарта

В настоящем стандарте работы, которые могут выполняться в жизненном цикле программных средств, распределены по пяти основным, восьми вспомогательным и четырем организационным процессам. Каждый процесс жизненного цикла разделен на набор работ; каждая работа разделена на набор задач. Все процессы жизненного цикла описаны ниже (рисунок 1.19).

Основные процессы жизненного цикла

Раздел 5 состоит из пяти процессов, которые реализуются под управлением основных сторон, вовлеченных в жизненный цикл программных средств. Под основной стороной понимают одну из тех организаций, которые инициируют или выполняют разработку, эксплуатацию или сопровождение программных продуктов. Основные стороны – заказчик, поставщик, разработчик, оператор и персонал сопровождения программных продуктов. Основными процессами являются:

1. Процесс заказа (подразд. 5.1). Определяет работы заказчика, т. е. организации, которая приобретает систему, программный продукт или программную услугу.

2. Процесс поставки (подразд. 5.2). Определяет работы поставщика, т. е. организации, которая поставляет систему, программный продукт или программную услугу заказчику.

3. Процесс разработки (подразд. 5.3). Определяет работы разработчика, т. е. организации, которая проектирует и разрабатывает программный продукт.

4. Процесс эксплуатации (подразд. 5.4). Определяет работы оператора, т. е. организации, которая обеспечивает эксплуатационное обслуживание вычислительной системы в заданных условиях в интересах пользователей.

5. Процесс сопровождения (подразд. 5.5). Определяет работы персонала сопровождения, т. е. организации, которая предоставляет услуги по сопровождению программного продукта, состоящие в контролируемом измененииипрограммного продукта с целью сохранения его исходного состояния и функциональных возможностей. Данный процесс охватывает перенос и снятие с эксплуатации программного продукта.

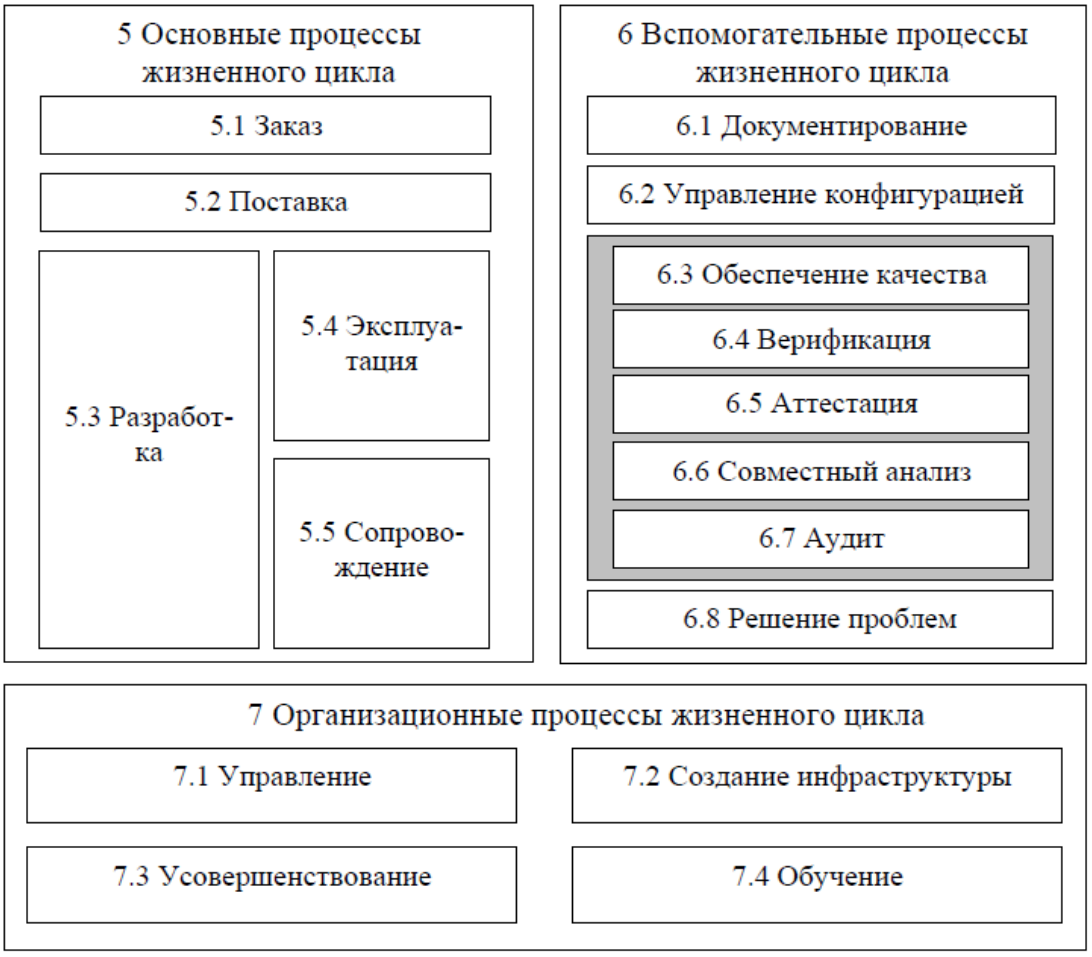


Рисунок 1.19 - Структура стандарта

Вспомогательные процессы жизненного цикла

Раздел 6 состоит из восьми процессов. Вспомогательный процесс является целенаправленной составной частью другого процесса, обеспечивающей успешную реализацию и качество выполнения программного проекта. Вспомогательный процесс, при необходимости, инициируется и используется другим процессом. Вспомогательными процессами являются:

1. Процесс документирования (подразд. 6.1). Определяет работы по описанию информации, выдаваемой в процессе жизненного цикла.
2. Процесс управления конфигурацией (подразд. 6.2). Определяет работы по управлению конфигурацией.
3. Процесс обеспечения качества (подразд. 6.3). Определяет работы по объективному обеспечению того, чтобы программные продукты и процессы соответствовали требованиям, установленным для них, и реализовывались в рамках утвержденных планов. Совместные анализы, аудиторские проверки, верификация и аттестация могут использоваться в качестве методов обеспечения качества.
4. Процесс верификации (подразд. 6.4). Определяет работы (заказчика, поставщика или независимой стороны) по верификации программных продуктов по мере реализации программного проекта.
5. Процесс аттестации (подразд. 6.5). Определяет работы (заказчика, поставщика или независимой стороны) по аттестации программных продуктов программного проекта.
6. Процесс совместного анализа (подразд. 6.6). Определяет работы по оценке состояния и результатов какой-либо работы. Данный процесс может использоваться двумя любыми сторонами, когда одна из сторон (проверяющая) проверяет другую сторону (проверяемую) на совместном совещании.
7. Процесс аудита (подразд. 6.7). Определяет работы по определению соответствия требованиям, планам и договору. Данный процесс может использоваться двумя сторонами, когда одна из сторон (проверяющая) контролирует программные продукты или работы другой стороны (проверяемой).
8. Процесс решения проблемы (подразд. 6.8). Определяет процесс анализа и устранения проблем (включая несоответствия), независимо от их характера и источника, которые были обнаружены во время осуществления разработки, эксплуатации, сопровождения или других процессов.

Организационные процессы жизненного цикла

Раздел 7 состоит из четырех процессов. Организационные процессы применяются в какой-либо организации для создания и реализации основной структуры, охватывающей взаимосвязанные процессы жизненного цикла и соответствующий персонал, а также для постоянного совершенствования данной структуры и процессов. Эти процессы, как правило, являются типовыми, независимо от области реализации конкретных проектов и договоров; однако уроки, извлеченные из таких проектов и договоров, способствуют совершенствованию организационных вопросов. Организационными процессами являются:

1. Процесс управления (подразд. 7.1). Определяет основные работы по управлению, включая управление проектом, при реализации процессов жизненного цикла.
2. Процесс создания инфраструктуры (подразд. 7.2). Определяет основные работы по созданию основной структуры процесса жизненного цикла.
3. Процесс усовершенствования (подразд. 7.3). Определяет основные работы, которые организация (заказчика, поставщика, разработчика, оператора, персонала сопровождения или администратора другого процесса) выполняет при создании, оценке, контроле и усовершенствовании выбранных процессов жизненного цикла.
4. Процесс обучения (подразд. 7.4). Определяет работы по соответствующему обучению персонала.

План конспекта:

1. Основные этапы развития технологии разработки.
2. Эволюция моделей жизненного цикла программного обеспечения.
3. Стандарты, регламентирующие процесс разработки программного обеспечения.

Работа с литературой:

Рекомендуемые источники информации

(№ источника)			
Основная	Дополнительная	Методическая	Интернет-ресурсы
1-3	1-3	1-3	1-2

Тема самостоятельного изучения № 2
Анализ проблемы и постановка задачи.

Вид деятельности студентов: самостоятельное изучение литературы
Итоговый продукт самостоятельной работы: конспект
Средства и технологии оценки: собеседование
Краткие теоретические сведения

2 АНАЛИЗ ПРОБЛЕМЫ И ПОСТАНОВКА ЗАДАЧИ

2.1 Введение в системный анализ

Системный анализ – система понятий, методов (среди которых должен быть метод декомпозиции) и технологий для изучения, описания, реализации систем различной природы и характера, междисциплинарных проблем; это система общих законов, методов, приемов исследования таких систем.

Любую предметную область также можно определить как системную.

Предметная область – раздел науки, изучающий предметные аспекты системных процессов и системные аспекты предметных процессов и явлений. Это определение можно считать системным определением предметной области.

Пример. Информатика – наука, изучающая информационные аспекты системных процессов и системные аспекты информационных процессов. Это определение можно считать системным определением информатики.

Системный анализ тесно связан с синергетикой.

Синергетика – междисциплинарная наука, изучающая общие идеи, методы и закономерности организации (изменения структуры, ее пространственно-временного усложнения) различных объектов и процессов, инварианты этих процессов. «Синергетика» в переводе с греческого означает «совместный», «согласованно действующий».

Системный анализ тесно связан и с философией. Философия дает общие методы содержательного анализа, а системный анализ дает общие методы формального, межпредметного анализа предметных областей, выявления и описания, изучения их системных инвариантов.

Можно дать и философское определение системного анализа: системный анализ – это прикладная диалектика.

Системный анализ предоставляет к использованию в различных науках, системах следующие методы и процедуры:

- абстрагирование и конкретизация;
- анализ и синтез;
- индукция и дедукция;
- формализация; структурирование; макетирование; алгоритмизация; моделирование; программное управление;
- распознавание, классификация и идентификация образов;
- экспертное оценивание и тестирование и другие методы и процедуры.

2.1.1 Системные ресурсы

Имеются следующие основные типы ресурсов в природе и в обществе.

Вещество – наиболее хорошо изученный ресурс, который в основном представлен таблицей Д. И. Менделеева и пополняется не так часто. Вещество выступает как отражение постоянства материи в природе, как мера однородности материи.

Энергия – не полностью изученный тип ресурсов, например, мы не владеем управляемой термоядерной реакцией. Энергия выступает как отражение изменчивости материи, переходов из одного вида в другой, как мера необратимости материи.

Информация – мало изученный тип ресурсов. Информация выступает как отражение порядка, структурированности материи, как мера порядка, самоорганизации материи (и

социума). Сейчас под этим термином мы будем понимать некоторые сообщения, ниже мы рассмотрим его более детально.

Человек – выступает как носитель интеллекта высшего уровня и является в экономическом, социальном, гуманитарном смысле важнейшим и уникальным ресурсом общества, выступает как мера разума, интеллекта и целенаправленного действия, мера социального начала, высшей формы отражения материи (сознания).

Организация (или организованность) выступает как форма ресурсов в социуме, группе, которая определяет его структуру, включая институты человеческого общества и его надстройки, выступает как мера упорядоченности ресурсов. Организация системы связана с наличием некоторых причинно-следственных связей в этой системе. Организация системы может иметь различные формы, например, биологическую, информационную, экологическую, экономическую, социальную, временную, пространственную и она определяется причинно-следственными связями в материи и социуме.

Пространство – мера протяженности материи (события), распределения ее (его) в окружающей среде.

Время – мера обратимости (необратимости) материи, событий. Время неразрывно связано с изменениями действительности.

Можно говорить о различных полях, в которые «помещен» любой человек: материальном, энергетическом, информационном, социальном, – их пространственных и временных характеристиках.

Пример. Рассмотрим простую задачу – пойти утром на занятия в вуз.

Эта часто решаемая студентом задача имеет все аспекты:

- материальный, физический аспект – студенту необходимо переместить некоторую массу, например, учебников и тетрадей на нужное расстояние;
- энергетический аспект – студенту необходимо иметь и затратить нужное количество энергии на перемещение;
- информационный аспект – необходима информация о маршруте движения и месторасположении вуза, кроме того, нужно обрабатывать инфор- мацию по пути своего движения;
- человеческий аспект – перемещение, в частности переезд на автобусе, невозможен без человека, например, без водителя автобуса;
- организационный аспект – необходимы подходящие транспортные сети и маршруты, остановки и т. д.;
- пространственный аспект – перемещение на определенное расстояние;
- временной аспект – на данное перемещение будет затрачено время (за которое произойдут соответствующие необратимые изменения в среде, в отношениях, в связях).

Все типы ресурсов тесно связаны. Более того, они невозможны друг без друга: актуализация одного из них ведет к актуализации другого.

Пример. При сжигании дров в печке выделяется тепловая энергия, тепловая энергия используется для приготовления пищи, пища используется для получения биологической энергии организма, биологическая энергия используется для получения информации (например, решения некоторой задачи), перемещения во времени и в пространстве. Человек и во время сна расходует свою биологическую энергию на поддержание информационных процессов в организме; более того, сон – продукт таких процессов.

Социальная организация и активность людей совершенствуют информационные ресурсы, процессы в обществе, последние, в свою очередь, совершенствуют производственные отношения.

Если классическое естествознание объясняет мир исходя из движения, взаимопревращений вещества и энергии, то сейчас реальный мир, объективная реальность могут быть объяснены лишь с учетом сопутствующих системных, особенно, системно-информационных процессов.

2.2 Анализ проблемы и моделирование предметной области с использованием системного подхода

2.2.1 Основные положения

Анализ проблем – это процесс осознания реальных проблем и потребностей пользователей и предложения решений, позволяющих удовлетворить эти потребности.

Цель анализа проблемы состоит в том, чтобы добиться лучшего понимания решаемой проблемы до начала разработки.

Чтобы выявить причины (или проблемы, стоящие за проблемой), необходимо опросить людей, которых непосредственно затрагивает данная проблема. Выявление актантов системы является ключевым шагом в анализе проблемы.

При этом необходимо проанализировать и понять область проблемы и исследовать разнообразные области решений. Как правило, возможных решений множество, и нам необходимо найти то, которое наиболее соответствует решаемой проблеме.

Чтобы иметь возможность провести анализ проблемы, полезно определить, что же собой представляет проблема. По определению Гауса и Вайнберга (Cause, Weinberg, 1989) проблема – это разница между желаемым и воспринимаемым.

Это определение достаточно разумно, по крайней мере, оно устраняет часто встречающееся среди разработчиков заблуждение, что подлинная проблема заключается в том, что пользователь не понимает, в чем состоит проблема! Согласно данному определению, если пользователь ощущает нечто как проблему, это и есть настоящая проблема, и она достойна внимания.

При анализе проблемы необходимо осуществить следующие пять этапов:

1) достигнуть соглашения об определении проблемы;
2) выделить основные причины – проблемы, стоящие за проблемой;
3) выявить заинтересованных лиц и пользователей;
4) определить границу системы решения;
5) выявить ограничения, которые необходимо наложить на решение.

2.2.2 Этап 1. Достижение соглашения об определении проблемы

Один из простейших способов заключается в том, чтобы просто записать проблему и выяснить, все ли согласны с такой постановкой (таблица 2.1).

В рамках этого процесса зачастую полезно рассмотреть преимущества предлагаемого решения, причем их следует описывать на языке клиентов/пользователей. Это обеспечивает дополнительную содержательную основу для понимания реальной проблемы. Рассматривая с точки зрения клиента эти преимущества, мы также достигаем лучшего понимания их взгляда на проблему в целом.

Таблица 2.1 - Стандартная форма постановки проблемы

Элемент	Описание
Проблема	[Описание проблемы]
Воздействует на	[Указание лиц, на которых оказывает влияние данная проблема]
Результатом чего является	[Описание воздействия данной проблемы на заинтересованных лиц и бизнес-деятельность]
Выигрыш от	[Указание предлагаемого решения]
Может состоять в следующем	[Список основных предоставляемых решением преимуществ]

2.2.3 Этап 2. Выделение основных причин, стоящих за проблемой

Для понимания реальной проблемы и ее причин можно использовать множество методов. Одним из них является метод анализа корневых причин, представляющий собой семантический способ нахождения причин, лежащих в основе рассматриваемой проблемы или ее проявления.

Рассмотрим реальный пример. Компания GoodsAreUs, занимающаяся торговлей по каталогу, производит и рассылает на дом множество недорогих товаров различных наименований. Решив заняться проблемой недостаточной прибыльности, компания использует рекомендуемую ей программой обеспечения качества методику «качество – во всем» (total quality management, TQM). Применив данный подход, компания практически сразу обратила внимание на ущерб

от несоответствия (cost of nonconformance), который представляет собой стоимость всего, что идет не так, как надо, и приводит к бесполезным затратам. Этот ущерб включает в себя переделки, остатки, неудовлетворенность клиента, текучесть кадров и другие негативные факторы. Проанализировав ущерб от несоответствия, компания заподозрила, что наибольший вклад в него вносят «остатки».

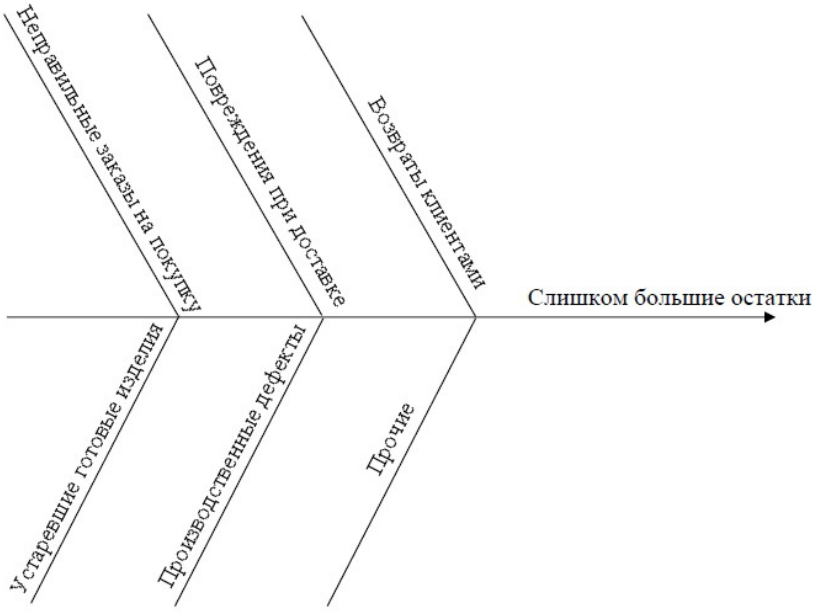


Рисунок 2.1 - Диаграмма в форме рыбного скелета для отображения корневых причин

Следующим шагом должно стать определение того, какие факторы оказывают влияние на величину остатков. TQM советует для обнаружения проблем, стоящих за проблемой, использовать диаграмму в форме рыбного скелета (рисунок 2.1). В нашем случае компания выявила много источников, вносящих свой вклад в остатки. Каждый источник указан как одна из «косточек» на диаграмме.

Способ выявления корневых причин зависит от конкретного случая.

Существует несколько способов выявления причин:

- опрос сотрудников, непосредственно занимающихся этим делом;
- «мозговой штурм» с участием тех, кто знаком с данной областью;
- метод упрощенной спецификации приложений;
- совместная разработка приложений;
- пользовательский сценарий и сеансы разработки схем выбора.

2.2.3.1 Устранение корневых причин

Качественные данные свидетельствуют, что многие корневые причины просто не стоят того, чтобы их устранять, поскольку затраты на их устранение превысят причиняемый проблемой ущерб.

Предложение заменить существующую систему или разработать новую, должно быть хорошо аргументированным. Для этого нужно предоставить стоимостное обоснование такой системы, оценив затраты на разработку и дивиденды от эксплуатации этой системы.

Продолжая анализ, можно применить новую диаграмму в виде рыбного скелета для определения того, какие типы ошибок вносят наибольший вклад в проблему неправильности заказов. Полученные данные можно затем использовать для определения функций новой системы программного обеспечения, которая призвана устранить эти ошибки. Раз мы приняли решение, что проблема неправильных заказов на покупку достойна решения, можно создать для нее формальную постановку (таблица 2.2).

Таблица 2.2 - Постановка проблемы ввода заказов на покупку

Элементы	Описание
Проблема воздействует на	Выполнение неправильных заказов на покупку, выполняющий заказы персонал, клиентов, производство, продажи и обслуживание клиентов
Результатом чего является	Увеличение остатков, повышение производственных затрат, неудовлетворенность клиентов, уменьшение прибыли
Выигрыш от	Новой системы, направленной на решение данной проблемы
Может состоять в следующем	Повышение точности заказов на покупку в точке ввода, совершенствование учета данных о покупках, в конечном счете – увеличение прибыли

После постановки проблемы, ее следует передать для ознакомления заказчикам и заинтересованным лицам, чтобы они внесли свои комментарии. Затем постановка проблемы доводится до сведения всех членов команды разработчиков с тем, чтобы все работали в направлении достижения общей цели.

2.2.4 Этап 3. Выявление заинтересованных лиц и пользователей

При решении любой сложной проблемы, как правило, приходится удовлетворять потребности различных групп заинтересованных лиц. Эти группы обычно имеют различные точки зрения на проблему и различные потребности, которые должны быть учтены в решении.

Заинтересованные лица – это все, на кого реализация новой системы или приложения может оказать материальное воздействие.

Понимание потребностей пользователей и других заинтересованных лиц является ключевым фактором в выработке успешного решения.

Первая категория заинтересованных лиц – это пользователи системы.

Их потребности легко учесть, поскольку они будут непосредственно привлекаться к определению и использованию системы. Вторую категорию составляют непрямые пользователи, а также те, на кого воздействуют только бизнес последствия разработки и те, кто воздействует на разработку. Потребности заинтересованных лиц, не являющихся пользователями, также необходимо выявить и учесть.

В зависимости от того, в какой предметной области работает команда, выявление заинтересованных лиц может оказаться как тривиальным, так и нетривиальным этапом анализа проблемы. Часто достаточно провести простой опрос среди тех, кто принимает решения, а также опросить потенциальных пользователей и другие заинтересованные стороны. В этом процессе могут оказаться полезными следующие вопросы.

Кто является пользователями системы?

Кто является заказчиком (экономическим покупателем) системы?

На кого еще окажут влияние результаты работы системы?

Кто будет оценивать и принимать систему, когда она будет представлена и развернута?

Существуют ли другие внутренние или внешние пользователи системы, чьи потребности необходимо учесть?

Кто будет заниматься сопровождением новой системы?

Не забыли ли мы кого-нибудь?

В нашем примере замены системы заказов на покупку основными и наиболее очевидными пользователями являются служащие, занимающиеся вводом заказов на покупку. Они определенно являются заинтересованными лицами, так как их производительность, удобство, комфорт, выполнение работы и ее результаты зависят от системы. Кого еще из заинтересованных лиц можно выделить?

На руководителя отдела приема заказов система также оказывает непосредственное воздействие, но он взаимодействует с системой не напрямую, а посредством различных интерфейсов пользователя и форм отчетов. Главный финансист компании также, очевидно, принадлежит к заинтересованным лицам, так как ожидается, что система повлияет на производительность, качество предоставляемых услуг и прибыльность компании. Наконец,

администратор информационной системы и члены команды, разрабатывающей приложение, также являются заинтересованными лицами, так как они будут отвечать за разработку и сопровождение системы. Они также, как и пользователи, будут зависеть от поведения системы. Результаты выявления пользователей и заинтересованных лиц новой системы ввода заказов на покупку представлены в таблица 2.3.

Таблица 2.3 - Пользователи и лица, заинтересованные в новой системе

Пользователи	Другие заинтересованные лица
Служащие, занимающиеся вводом заказов Руководитель отдела приема заказов Контроль производства Служащий, выписывающий счета	Администратор информационной системы и команда разработчиков Главный финансист Управляющий производством

2.2.5 Этап 4. Определение границ системы-решения

После того как согласована постановка проблемы и выявлены пользователи и заинтересованные лица, можно перейти к *определению системы,* разрабатываемой для решения данной проблемы. Это важный момент, когда необходимо постоянно помнить как о понимании проблемы, так и о свойствах потенциального решения.

Граница системы описывает оболочку, в которой заключена система (рисунок 2.2). Информация в виде ввода и вывода передается от находящихся вне системы пользователей системе и обратно. Все взаимодействия с системой осуществляются посредством интерфейсов между системой и внешним миром.

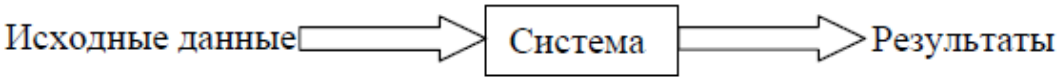


Рисунок 2.2 - Отношение ввод/система/вывод

Другими словами, если мы собираемся нечто создать или модифицировать – это часть нашего решения, которая находится внутри границы; если нет – это нечто внешнее по отношению к системе. Таким образом, *мы делим мир на два интересующих нас класса*.

1. Наша система.
2. То, что взаимодействует с нашей системой.

Определим то, что взаимодействует с нашей системой, общим понятием «актанты» (actors). Они выполняют некоторые действия, заставляя систему делать ее работу. Актант изображается простой пиктограммой в виде человечка. Его определение выглядит следующим образом.

Актант – это находящееся вне системы нечто (или некто), взаимодействующее с системой.

С помощью данного понятия мы можем проиллюстрировать границы системы.

Во многих случаях границы системы очевидны. Например, однопользовательский персональный планировщик контактов, работающий на автономной платформе Windows 2000, имеет достаточно хорошо определенные границы. Имеется всего один пользователь и одна платформа. Интерфейсы между пользователем и приложением состоят из диалогов, посредством которых пользователь получает доступ к информации системы, и неких выходных сообщений и коммуникационных путей, которые система использует для документирования или передачи этой информации.

Для системы ввода заказов из нашего примера, которая должна быть объединена с уже существующей информационной системой компании, границы не столь очевидны. Аналитик должен определить, будут ли данные использоваться совместно с другими приложениями, должно ли новое приложение распределяться по разным хостам и клиентам, а также кто будет пользователем. Например, должен ли персонал, занятый в производстве, иметь интерактивный доступ к заказам на покупку? Обеспечивается ли контроль качества или функции аудита? Будет ли система выполняться на компьютере-мэйнфрейме или на новом компьютере-клиенте? Должны ли предоставляться специальные отчеты?

Выявление актантов является ключевым аналитическим этапом в анализе проблемы.

Ответы на следующие вопросы помогут их обнаружить.

Кто будет поставлять, использовать или удалять информацию из системы?

Кто будет управлять системой?

Кто будет осуществлять сопровождение системы?

Где будет использоваться система?

Откуда система получает информацию?

Какие внешние системы будут взаимодействовать с системой?

Имея ответы на эти вопросы, аналитик может создать блок-схему, описывающую границы системы, пользователей и другие интерфейсы.

2.2.6. Этап 5. Выявление ограничений, налагаемых на решение

Ограничение уменьшает степень свободы, которой мы располагаем при предложении решения.

Каждое ограничение может значительно сузить нашу возможность создать предполагаемое решение. Следовательно, в процессе планирования необходимо тщательно изучить все ограничения. Многие из них могут даже заставить нас пересмотреть изначально предполагавшийся технологический подход.

Необходимо учитывать, что существуют различные источники ограничений (экономические, технические, политические и т. д.). Ограничения могут быть заданы еще до начала работы («Никакой новой аппаратуры!»), но может получиться, что нам действительно придется их выявлять.

Чтобы их выявить, полезно знать, на что следует обратить внимание.

В таблице 2.4 указаны возможные источники системных ограничений. Приведены в таблице вопросы помогут выявить большую часть ограничений. Часто полезно получить объяснение ограничения, как для того, чтобы убедиться, что вы поняли его назначение, так и для того, чтобы можно было обнаружить (если такое произойдет), что данное ограничение больше не применимо к вашему решению.

После того как ограничения выявлены, некоторые из них станут требованиями к новой системе. Другие ограничения будут оказывать влияние на ресурсы и планы реализации. Именно при анализе проблемы необходимо выявить потенциальные источники ограничений и понять, какое влияние каждое ограничение окажет на область возможных решений.

Возвратимся к нашему примеру. Ограничения, которые могут налагаться на новую систему ввода заказов на покупку, представлены в таблице 2.5.

Таблица 2.4 - Возможные источники ограничений системы

Источник	Образцы вопросов
Экономический	Какие финансовые или бюджетные ограничения следует учесть?
	Существуют ли соображения, касающиеся себестоимости и ценообразования?
	Существуют ли вопросы лицензирования?
Политический	Существуют ли внешние или внутренние политические вопросы, влияющие на потенциальное решение?
	Существуют ли проблемы в отношениях между подразделениями?
Технический	Существуют ли ограничения в выборе технологий?
	Должны ли мы работать в рамках существующих платформ или технологий?
	Запрещено ли использование любых новых технологий?
	Должны ли мы использовать какие-либо закупаемые пакеты программного обеспечения?
Системный	Будет ли решение создаваться для наших существующих систем?
	Должны ли мы обеспечивать совместимость с существующими решениями?
	Какие операционные системы и среды должны поддерживаться?
Эксплуатационный	Существуют ли ограничения информационной среды или правовые ограничения?
	Юридические ограничения?
	Требования безопасности?
	Какими другими стандартами мы ограничены?
График и ресурсы	Определен ли график?
	Ограничены ли мы существующими ресурсами?
	Можем ли мы привлекать работников со стороны?
	Можем ли мы увеличить штат? Временно? Постоянно?

Таблица 2.5 - Ограничения, налагаемые на систему ввода заказов на покупку

Источник	Ограничение	Объяснение
Эксплуатационный	Копия данных заказа на покупку должна оставаться в унаследованной базе данных в течение одного года	Риск потери данных слишком велик
Системы и операционные системы	Приложение должно занимать на сервере не более 20 мегабайт	Количество доступной памяти сервера ограничено
Средства, выделенные на оборудование	Система должна быть разработана на существующем сервере	Сокращение издержек и поддержка существующих систем
Средства, выделенные на оплату труда персонала	Фиксированный штат; не привлекать работников со стороны	Фиксированные расходы на зарплату по отношению к текущему бюджету
Технические требования	Должна использоваться новая объектно-ориентирован-ная технология	Мы надеемся, что эта технология повысит производительность и надежность ПО

2.3 Методология ARIS

В теории систем различают структуру системы и ее поведение. Структура описывает статику системы, а поведение описывает динамику. Четыре первых представления в ARIS (организационное, функциональное, представление выходов и представление данных) описывают структуру системы. В процессном представлении устанавливаются связи перечисленных представлений, и описывается динамическое поведение системы. На рисунке 2.3 перечисленные представления сгруппированы в пять частей, образующих здание АРИС.

Функциональное представление содержит описание целей, выполняемых функций, перечень отдельных подфункций, а также общие взаимосвязи и связи подчиненности, которые существуют между функциями, целями и факторами успеха.

Организационное представление описывает взаимодействие пользователей и организационных единиц (ОЕ), а также их связи и имеющие к ним отношение структуры. Организационные модели служат для описания иерархической структуры организации, где группируются субъекты ответственности, ОЕ, должностей и средств, выполняющих работу над объектами, а также для многого другого.

В представлении данных описывают информационную среду предприятия (среду обработки данных), события, сообщения и т. д., где неявно фиксируются также объекты в виде информационных услуг.

Представление процессов (управления) используется для описания связей между тремя предшествующими представлениями. Интеграция этих связей в пределах отдельного представления делает возможным учесть все связи без избыточности. Модели представления процессов описывают отношения между отдельными моделями в рамках всего бизнес-процесса. Это позволяет отслеживать все двусторонние и многосторонние отношения между моделями различных видов, а также полностью описать бизнес-процесс.

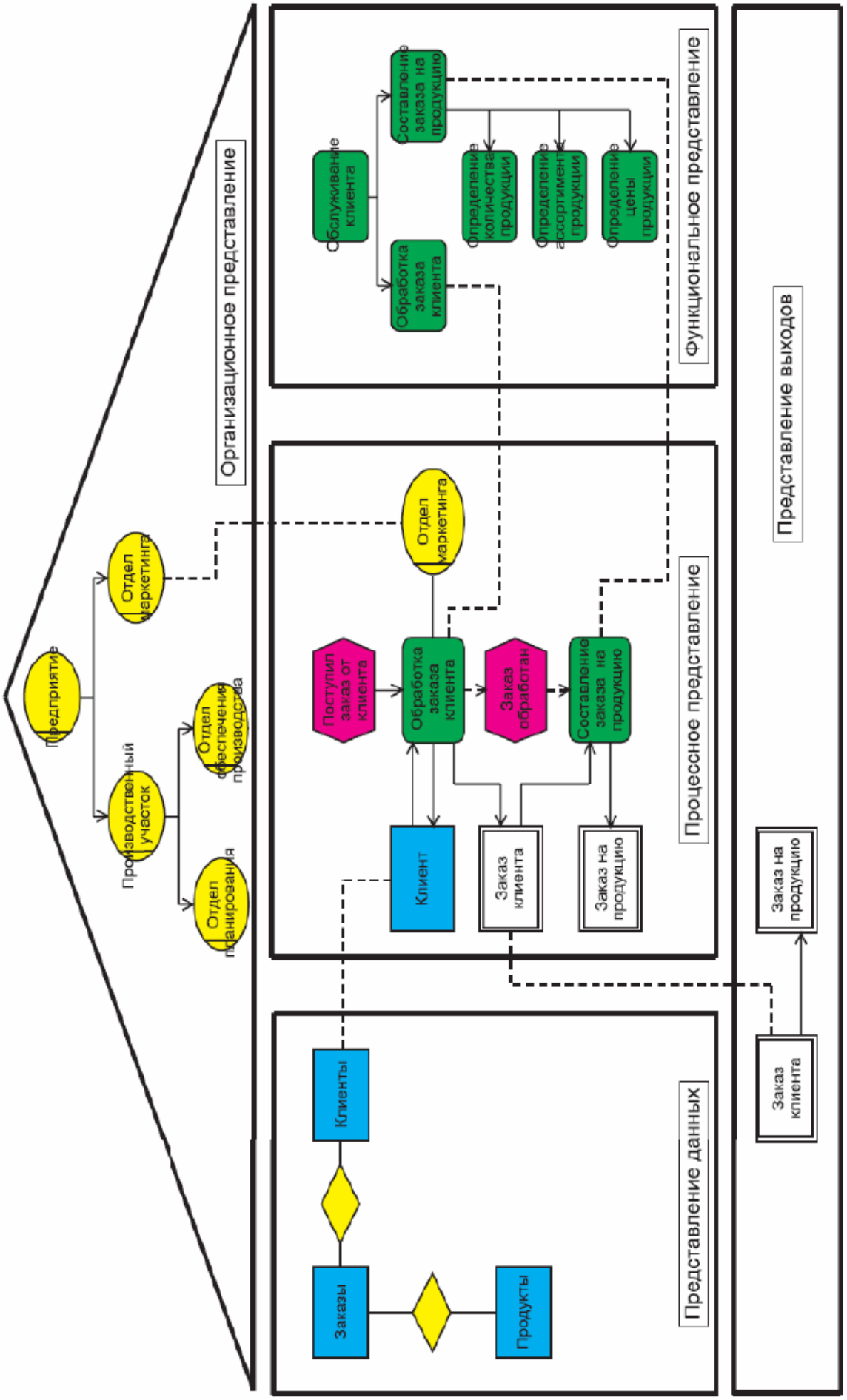


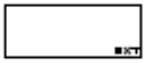
Рисунок 2.3. «Здание» ARIS. Представления

2.3.1 Организационная модель

Организационная диаграмма позволяет всесторонне описать организационно-штатную структуру предприятия. Организационные диаграммы показывают имеющиеся организационные подразделения (как исполнители функций) и их взаимозависимости в соответствии с выбранными критериями структурирования. Отдельные организационные единицы соединяются связями для указания иерархии.

Модель организационной структуры предприятия может содержать следующие структурные элементы: организационная единица, должность или позиция, личность, группа и др. (таблица 2.6).

Таблица 2.6 - Элементы организационной диаграммы

№ п/п	Графическая нотация	Наименование	Описание
1		Организационная единица (Organizational unit)	Элемент организационной структуры (структурное подразделение), который отвечает за выполнение определенных задач и преследует определенные цели
2		Позиция или должность (Position)	Наиболее мелкий организационный элемент на предприятии. Обязанности и административные полномочия такого элемента задаются должностными инструкциями
3		Группа (Group)	Несколько сотрудников, которые вместе работают над конкретной задачей в определенный период времени (например, группа проектирования)
4		Внешний сотрудник (External person)	Лицо, выполняющее определенные функции в компании, но не являющееся служащим этой компании. Он может назначаться к организационным единицам или функциям, к выполнению которых он привлекается со стороны или за которые он отвечает
5		Сотрудник (Internal person)	Является служащим компании и может назначаться к организационным единицам или функциям, которые он выполняет или за которые несет ответственность
6		Тип сотрудника (Person type)	Обобщает индивидуальных лиц, имеющих одинаковые характеристики. Эти характеристики могут иметь отношение, например, к одним и тем же полномочиям. Начальники отделов или бригадиры, например, должны следовать определенным правилам и выполнять определенные обязанности, которые достаточно описать только один раз
7		Местонахождение (Location)	Определяет физическое положение (местонахождение) организационных единиц, рабочих мест, образцов аппаратных компонентов и технических ресурсов компании. Местонахождение может ссылаться на область, город, предприятие, здание и т. д.

При построении модели организационной структуры между структурными элементами могут устанавливаться следующие типы связей: имеет в подчинении (Is superior), связан с (Is assigned to), принадлежит (Belongs to), является организационным управляющим (Is Organization Manager for), имеет в своем составе (Has member), состоит из (Is composed of), располагает (Is located at), взаимодействует с (Cooperates with), содержит (Subsumes), занимает (Occupies).

Пример организационной диаграммы приведен на рисунке 2.4.

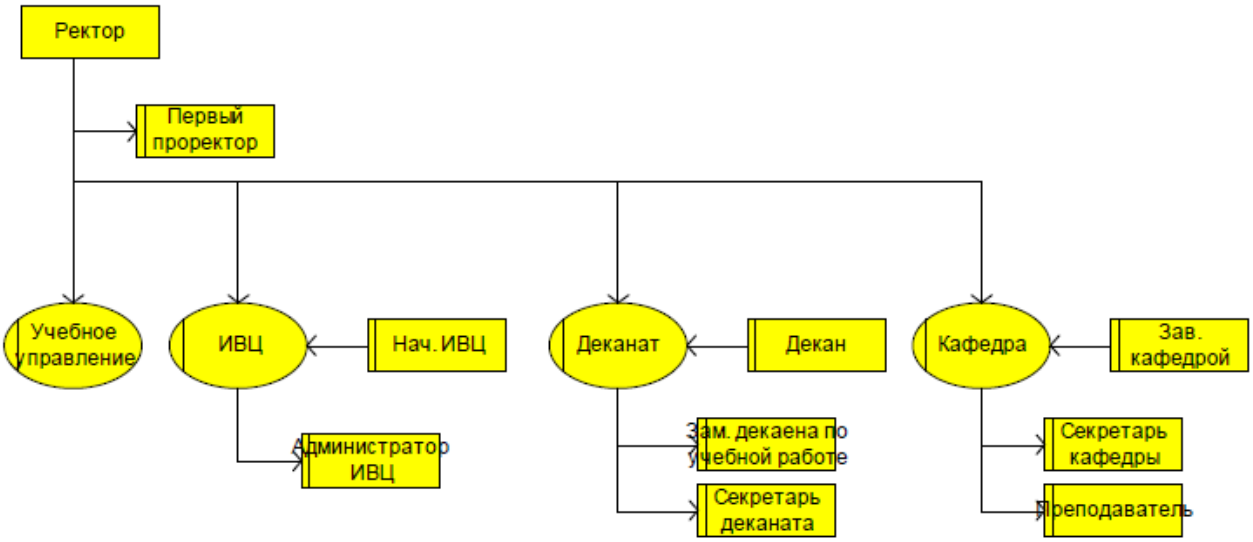


Рисунок 2.4 - Пример организационной диаграммы

2.3.2 Диаграмма цепочки добавленного качества

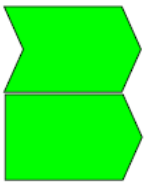
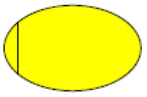
Для уменьшения сложности описания деятельности организации необходимо разработать иерархию моделей бизнес-процессов (БП) организации начиная с самого верхнего уровня и до моделей отдельных БП на нижнем уровне. Для описания модели верхнего уровня используется диаграмма типа Value-added chain diagram (VAD), название которой можно перевести как Модель цепочки добавленного качества (стоимости).

Диаграмма цепочек добавленного качества описывает функции организации, которые непосредственно влияют на реальный выход ее продукции. Эти функции создают последовательность действий, формируя добавленные значения: стоимость, количество, качество и т. д.

Диаграмма цепочек добавленной стоимости может содержать следующие структурные элементы: организационную единицу, должность или позицию, функцию и др. (таблица 2.7).

Пример диаграммы цепочек добавленной стоимости представлен на рисунок 2.5. Маленьким значком в нижнем правом углу элемента модели помечаются те функции, для которых существуют модели, раскрывающие эту функцию. Таким механизмом реализуется принцип декомпозиции.

Таблица 2.7 - Элементы диаграммы цепочки добавленного качества

№ п/п	Графическая нотация	Наименование	Описание
1		Функция	Действие или набор действий, выполняемых над исходным объектом (документом, материалом и т. п.) с целью получения заданного результата
2		Технический термин	
3		Организационная единица (Organizational unit)	Элемент организационной структуры (структурное подразделение), который отвечает за выполнение определенных задач и преследует определенные цели
4		Кластер (экземпляр)	Экземпляр объекта данных. Он представляет собой логический взгляд на набор объектов данных или структур

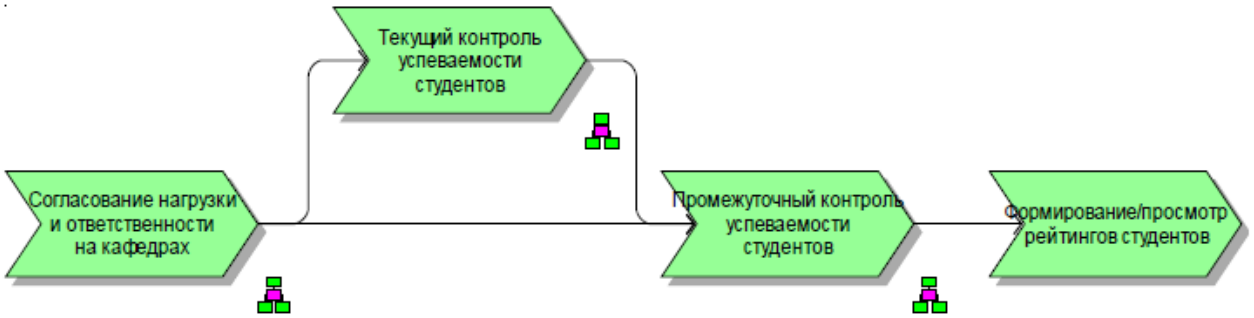


Рисунок 2.5 - Основные процессы учета успеваемости студентов в системе кредитов

2.3.3 Модели eEPC

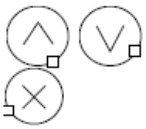
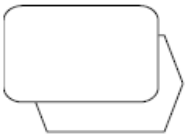
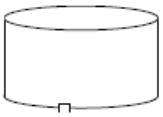
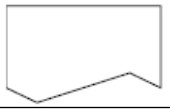
Цепочка процесса, управляемая событиями, – Extended event driven process chain (eEPC).

С помощью диаграмм eEPC процедуры БП представляются как логические последовательности событий. События определяют, какое состояние или отношение будет переключать функцию и какое состояние наступит после ее выполнения. Поэтому модель eEPC должна всегда иметь запускающие и завершающие события.

Одно событие может инициировать выполнение одновременно нескольких функций, и, наоборот, функция может быть результатом наступления нескольких событий. Объединения нескольких событий или функций отображаются на диаграмме eEPC с помощью соединителей в виде небольшого кружка. Эти соединители не только отображают графические связи между объектами модели, но и определяют логические связи между ними. Различают два типа связи логических операторов – **связи событий и связи функций**.

Диаграмма цепочек, управляемых событиями, кроме элементов организационной диаграммы и диаграммы данных, может содержать следующие структурные элементы: функцию, событие и др. (таблица 2.8).

Таблица 2.8 - Элементы диаграммы цепочек, управляемых событиями

№ п/п	Графическая нотация	Наименование	Описание
1		Функция	Действие или набор действий, выполняемых над исходным объектом (документом, материалом и т. п.) с целью получения заданного результата
2		Событие	Состояние, которое является существенным для управления БП и которое оказывает влияние или контролирует дальнейшее развитие одного или более БП. Изменения состояния отражаются с помощью информационных объектов
3		Правило	Представляет собой правило разветвления и слияния веток БП. Если перейти к рассмотрению каждой отдельной функции БП, то можно сказать, что правило отражает логическое соотношение между несколькими исходными для функции событиями и несколькими результирующими
4		Интерфейс функции	Используется для обозначения функции внешнего БП, который либо не описывается в данной модели, либо является БП другой предметной области
5		Носитель информации	Представляет собой средство хранения информации. Оно может быть реализовано, к примеру, в виде картотеки или компьютерных файлов или БД
6		Документ	Обозначает любой вид документов

Пример диаграммы цепочек, управляемых событиями, представлен на рисунок 2.6.

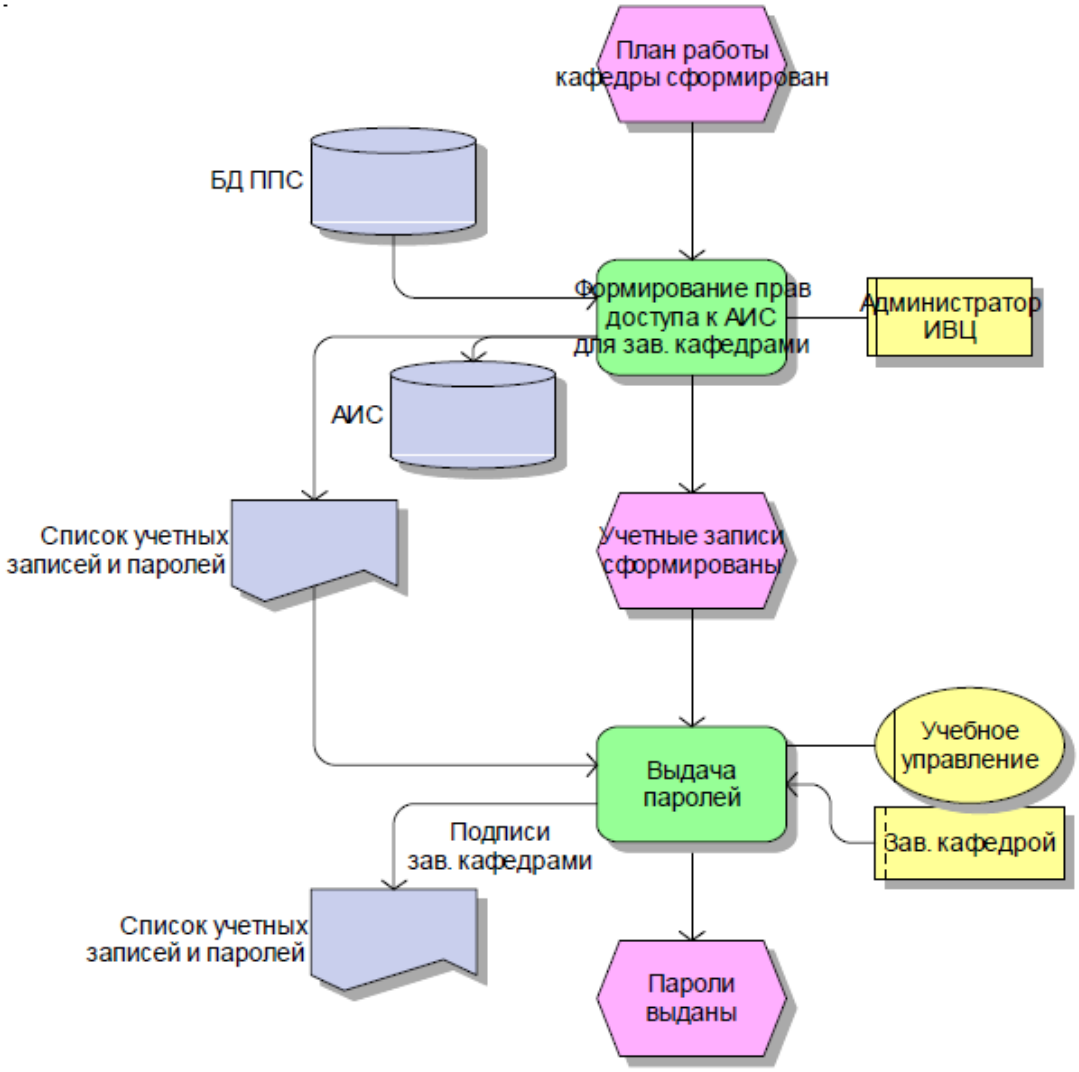


Рисунок 2.6 - Согласование нагрузки и ответственности на кафедрах

2.4 Стандарты IDEF0–IDEF3

2.4.1 Методология описания бизнес-процессов IDEF3

IDEF3 – способ описания процессов, основной целью которого является обеспечение структурированного метода, с помощью которого эксперт в предметной области может описать положение вещей как упорядоченную последовательность событий с одновременным описанием объектов, имею- щих непосредственное отношение к процессу.

Технология IDEF3 хорошо приспособлена для сбора данных, требующихся для проведения структурного анализа системы. В отличие от большинства технологий моделирования бизнес-процессов, IDEF3 не имеет жестких синтаксических или семантических ограничений, делающих неудобным описание неполных или нецелостных систем. Кроме того, автор модели (системный аналитик) избавлен от необходимости смешивать свои собственные предположения о функционировании системы с экспертными утверждениями в целях заполнения пробелов в описании предметной области. На рисунок 2.7 изображен пример описания процесса с использованием методологии IDEF3.

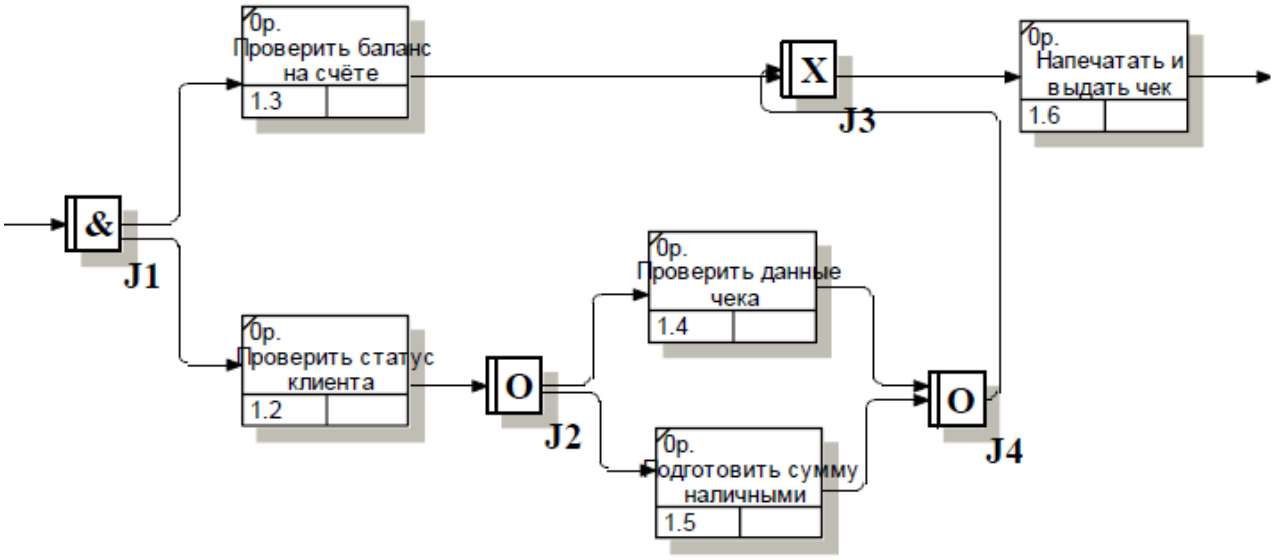


Рисунок 2.7 - Описание процесса в методологии IDEF3

Технология IDEF3 также может быть использована как метод проектирования бизнес-процессов. IDEF3-моделирование органично дополняет традиционное моделирование с использованием стандарта IDEF0. В настоящее время оно получает все большее распространение как вполне жизнеспособный путь построения моделей проектируемых систем для дальнейшего анализа имитационными методами. Имитационное тестирование часто используют для оценки эксплуатационных качеств разрабатываемой системы, более подробно методы имитационного анализа будут рассмотрены позднее.

2.4.1.1 Синтаксис и семантика моделей IDEF

Модели IDEF3. Основой модели IDEF3 служит так называемый сценарий бизнес-процесса, который выделяет последовательность действий или подпроцессов анализируемой системы. Поскольку сценарий определяет назначение и границы модели, довольно важным является подбор подходящего наименования для обозначения действий. Для подбора необходимого имени применяются стандартные рекомендации по предпочтительному использованию глаголов и отглагольных существительных. Например, «Обработать заказ клиента» или «Применить новый дизайн» – вполне подходящие названия сценариев.

Точка зрения для большинства моделей должна быть явным образом документирована. Обычно это название набора должностных обязанностей человека, являющегося источником информации о моделируемом процессе.

Для системного аналитика также важно понимание цели моделирования – набора вопросов, ответами на которые будет служить модель, границ моделирования (какие части системы войдут в модель, а какие не будут в ней отображены) и целевой аудитории (для кого разрабатывается модель).

Диаграммы. Главной организационной единицей модели IDEF3 является диаграмма. Взаимная организация диаграмм внутри модели IDEF3 особенно важна в случае, когда модель заведомо создается для последующего опубликования или рецензирования, что является вполне обычной практикой при проектировании новых систем. В этом случае системный аналитик должен позаботиться о таком информационном наполнении диаграмм, чтобы каждая из них была самодостаточной и в то же время понятной читателю.

Единица работы. Действие. Аналогично другим технологиям моделирования действие, или в терминах IDEF3 «единица работы» (Unit of Work – UOW), – другой важный компонент модели. Диаграммы IDEF3 отображают действие в виде прямоугольника. Как уже отмечалось, действия именуются с использованием глаголов или отглагольных существительных, каждому из действий присваивается уникальный идентификационный номер. Этот номер не используется вновь даже в том случае, если в процессе построения модели действие удаляется. В

диаграммах IDEF3 номер действия обычно предваряется номером его родителя (рисунок 2.8).

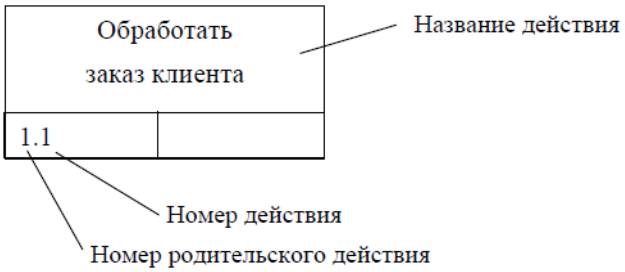


Рисунок 2.8 - Изображение и нумерация действия в диаграмме IDEF3

Связи. Связи выделяют существенные взаимоотношения между действиями. Все связи в IDEF3 являются однонаправленными, и, хотя стрелка может начинаться или заканчиваться на любой стороне блока, обозначающего действие, диаграммы IDEF3 обычно организовываются слева направо таким образом, что стрелки начинаются на правой и заканчиваются на левой стороне блоков. В таблице 2.9 приведены три возможных типа связей.

Таблица 2.9 - Типы связей в модели IDEF3

Изображение	Название	Назначение
→	Временное предшествование (Temporal precedence)	Исходное действие должно завершиться прежде, чем конечное действие сможет начаться
→→	Объектный поток (Object flow)	Выход исходного действия является входом конечного действия. Из этого, в частности, следует, что исходное действие должно завершиться прежде, чем конечное действие сможет начаться
- - - - →	Нечеткое отношение (Relationship)	Вид взаимодействия между исходным и конечным действиями задается аналитиком отдельно для каждого случая использования такого отношения

Связь типа «временное предшествование». Как видно из названия, связи этого типа отражают, что исходное действие должно полностью завершиться, прежде чем начнется выполнение конечного действия. Связь должна быть поименована таким образом, чтобы человеку, просматривающему модель, была понятна причина ее появления. Во многих случаях завершение одного действия инициирует начало выполнения другого (рисунок 2.9). В этом примере автор должен принять рекомендации рецензентов, прежде чем начать вносить соответствующие изменения в работу.

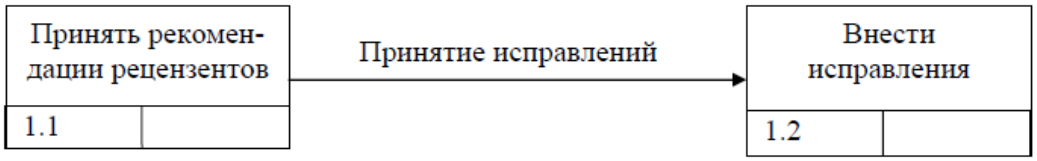


Рисунок 2.9 - Связь типа предшествование между действиями 1.1 и 1.2

Связь типа «объектный поток». Одной из наиболее часто встречающихся причин использования связи типа «объектный поток» состоит в том, что некоторый объект, являющийся результатом выполнения исходного действия, необходим для выполнения конечного действия. Такая связь отличается от связи временного предшествования двойным концом обозначающей ее стрелки. Наименования потоковых связей должны четко идентифицировать объект, который передается с их помощью. Временная семантика объектных связей аналогична связям предшествования. Это означает, что порождающее объектную связь исходное действие должно завершиться, прежде чем конечное действие начнет выполняться (рисунок 2.10). В приведенном примере счет на оплату услуг является результатом выполнения действия 1.1. Счет

необходим для проведения оплаты услуг.

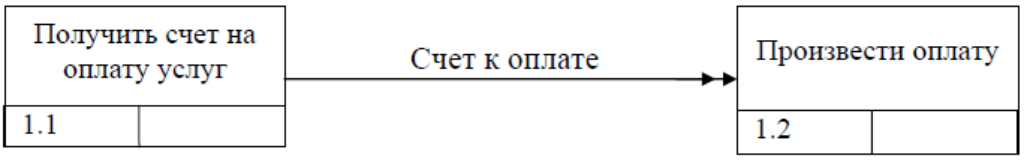


Рисунок 2.10 - Объектная связь между действиями 1.1 и 1.2

Связь типа «нечеткое отношение». Связи этого типа используются для выделения отношений между действиями, которые невозможно описать с использованием предшественных или объектных связей. Значение каждой такой связи должно быть определено, поскольку связи типа нечеткое отношение сами по себе не предполагают никаких ограничений. Одно из применений нечетких отношений – отображение взаимоотношений между параллельно выполняющимися действиями. Рисунок 2.11 иллюстрирует фрагмент процесса запуска бензопилы с водяным охлаждением и нечеткое отношение между действиями запустить двигатель и запустить водяной насос. Название стрелки может быть использовано для описания природы отношения, более подробное объяснение может быть приведено в виде отдельной ссылки.

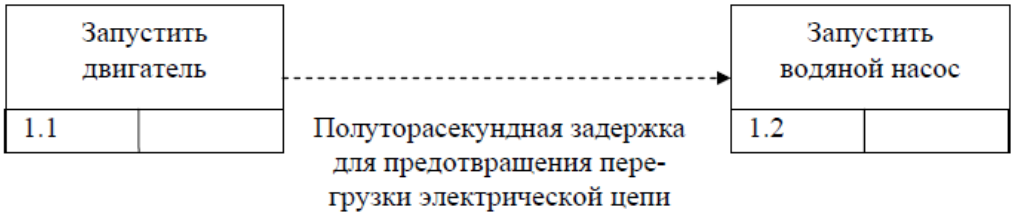


Рисунок 2.11 - Связь типа «нечеткое отношение»

Наиболее часто нечеткие отношения используются для описания специальных случаев связей предшествования, например для описания альтернативных вариантов временного предшествования. Альтернативная предшественной связи (рисунок 2.9) связь нечеткого отношения представлена на рисунок 2.12. В этом примере внесение исправлений начинается по мере получения замечаний от рецензентов, т. е. до непосредственного окончания действия по принятию замечаний.

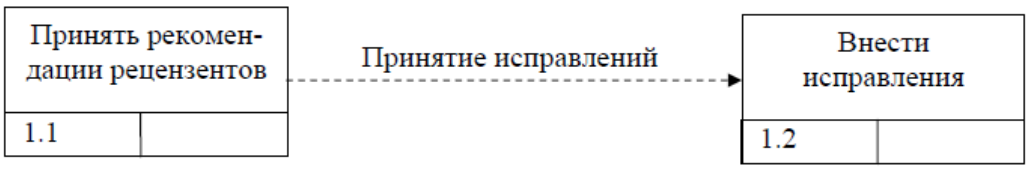


Рисунок 2.12 - Альтернатива связи предшествования

Соединения. Завершение одного действия может инициировать начало выполнения сразу нескольких других действий, или, наоборот, определенное действие может требовать завершения нескольких других действий для начала своего выполнения. Соединения разбивают или соединяют внутренние потоки и используют для описания ветвления процесса.

Разворачивающие соединения используются для разбиения потока. Завершение одного действия вызывает начало выполнения нескольких других.

Сворачивающие соединения объединяют потоки. Завершение одного или нескольких действий вызывает начало выполнения только одного другого действия. В таблице 2.10 приведены три типа соединений.

Таблица 2.10 - Типы соединений в модели IDEF3

Графическое обозначение	Название	Вид	Правила инициации
&	Соединение «И»	Разворачивающее	Каждое конечное действие обязательно инициируется
		Сворачивающее	Каждое исходное действие обязательно должно завер-
X	Соединение «Исключающее ИЛИ»	Разворачивающее	Одно и только одно конечное действие инициируется
		Сворачивающее	Одно и только одно исходное действие должно за-
O	Соединение «ИЛИ»	Разворачивающее	Одно (или более) конечное действие инициируется
		Сворачивающее	Одно (или более) исходное действие должно завер-

Примеры разворачивающих и сворачивающих соединений приведены на рисунке 2.13.

«*И*»-*соединения*. Соединения этого типа инициируют выполнение всех своих конечных действий. Все действия, присоединенные к сворачивающему «И»-соединению, должны завершиться, прежде чем может начать выполняться следующее действие. После обнаружения пожара инициируются включение пожарной сигнализации (рисунок 2.14), вызов пожарной охраны и начинается тушение пожара. Запись в журнал производится только тогда, когда все три перечисленных действия завершены.

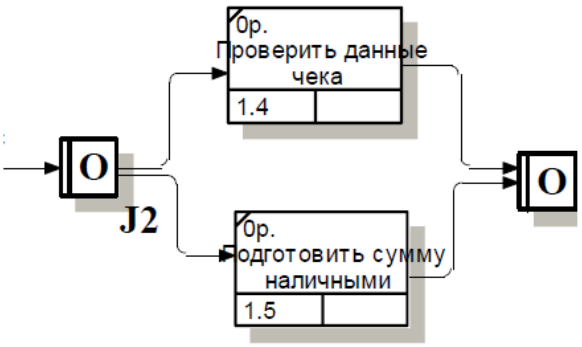


Рисунок 2.13 - Два вида соединений

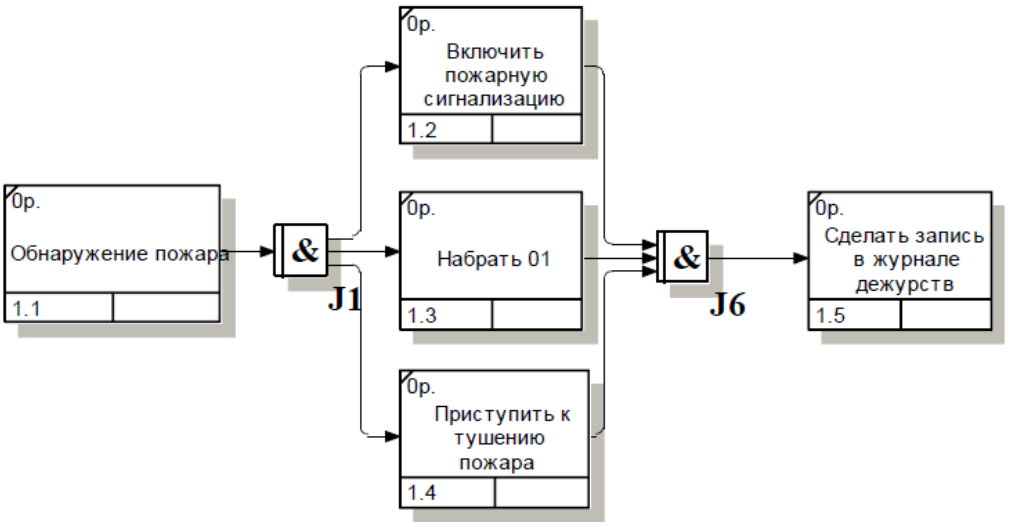


Рисунок 2.14 - «И»-соединения

Соединение «Исключающее ИЛИ». Вне зависимости от количества действий, прицепленных к сворачивающему или разворачивающему соединению «Исключающее ИЛИ», инициировано будет только одно из них, и поэтому только одно из них будет завершено перед тем, как любое действие, следующее за сворачивающим соединением «Исключающее ИЛИ», сможет начаться.

Если правила активации соединения известны, они обязательно должны быть документированы либо в его описании, либо пометкой стрелок, исходящих из разворачивающего соединения (рисунок 2.15).

Соединение «Исключающее ИЛИ» используется для отображения того факта, что студент не может одновременно быть направлен на лекции по двум разным курсам.

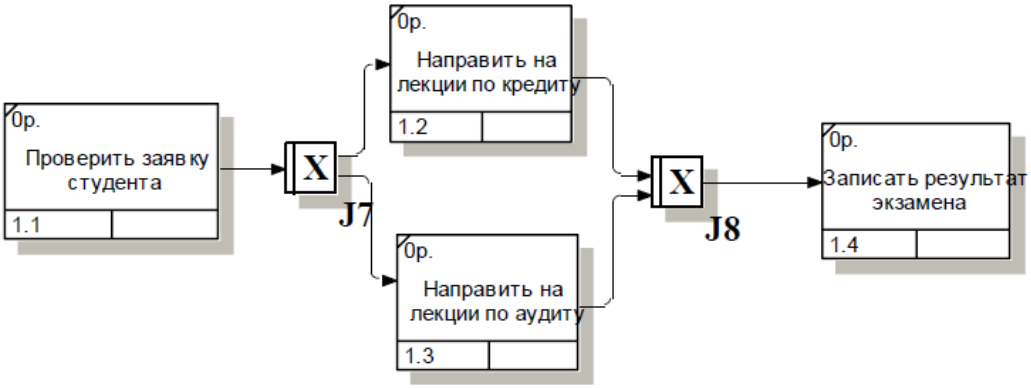


Рисунок 2.15 - Соединение «Исключающее ИЛИ»

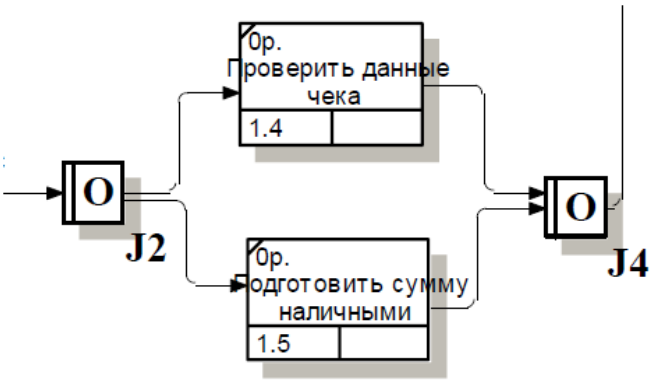


Рисунок 2.16 - Соединение «ИЛИ»

Соединение «ИЛИ». Соединения этого типа предназначены для описания ситуаций,

которые не могут быть описаны двумя предыдущими типами соединений. Аналогично связи нечеткого отношения соединение «ИЛИ» в основном определяется и описывается непосредственно системным аналитиком. Соединение J2 может активировать проверку данных чека и (или) проверку суммы наличных (рисунок 2.16). Проверка чека инициируется, если покупатель желает расплатиться чеком, проверка суммы наличных – при оплате наличными. Оба действия инициируются при частичной оплате чеком и частичной – наличными.

Декомпозиция действий. Действия в IDEF3 могут быть декомпозированы или разложены на составляющие, для более детального анализа. Декомпозировать действие можно несколько раз. Это позволяет документировать альтернативные потоки процесса в одной модели.

Для корректной идентификации действий в модели с множественными декомпозициями схема нумерации действий расширяется и наряду с номерами действия и его родителя включает в себя порядковый номер декомпозиции. Например, в номере действия 1.2.5: 1 – номер родительского действия, 2 – номер декомпозиции, 5 – номер действия.

2.4.1.2 Требования IDEF3 к описанию бизнес-процессов

Здесь мы рассмотрим построение IDEF3-диаграммы на основе выраженного в текстовом виде описания процесса. Предполагается, что в построении диаграммы принимают участие ее автор (в основном как системный аналитик) и один или несколько экспертов предметной области, которые будут представлять нам описание процесса.

Определение сценария, границ моделирования, точки зрения. Перед тем как попросить экспертов предметной области подготовить описание моделируемого процесса, должны быть документированы границы моделирования, чтобы экспертам была понятна необходимая глубина и полнота требуемого от них описания. Кроме того, если точка зрения аналитика на процесс отличается от обычной точки зрения для эксперта, это должно быть ясно и аккуратно описано.

Вполне возможно, что эксперты не смогут сделать приемлемое описание без применения формального опроса автором модели. В таком случае автор должен заранее приготовить набор вопросов таким же образом, как журналист заранее подготавливает вопросы для интервью.

Определение действий и объектов. Результатом работы экспертов обычно является текстовый документ, описывающий интересующий аналитика круг вопросов. В дополнение к нему может иметься письменная документация, позволяющая пролить свет на природу изучаемого процесса. Вне зависимости от того, является ли информация текстовой или вербальной, она анализируется и разделяется частями речи для идентификации списка действий (глаголы и отглагольные существительные), составляющих процесс, и объектов (имена существительные), участвующих в процессе.

В некоторых случаях возможно создание графической модели процесса в присутствии экспертов. Такая модель также может быть разработана после сбора всей необходимой информации, что позволяет не отнимать время экспертов на детали форматирования получающихся диаграмм.

Поскольку модели IDEF3 могут одновременно разрабатываться несколькими командами, IDEF3 поддерживает простую схему резервирования номеров действий в модели. Каждому аналитику выделяется уникальный диапазон номеров действий, что обеспечивает их независимость друг от друга. В таблице 2.11 номера действий выделяются каждому аналитику большими блоками. В этом примере Иван исчерпал данный ему вначале диапазон номеров и дополнительно получил второй.

Таблица 2.11 - Распределение диапазонов номеров IDEF3 между аналитиками

Аналитик	Диапазон номеров IDEF3
Иван	1–99
Петр	100–199
Николай	200–299
Иван	300–399

Последовательность и параллельность. Если модель создается после проведения

интервью, аналитик должен принять решения по построению иерархии участвующих в модели диаграмм, например, насколько подробно будет детализироваться каждая отдельно взятая диаграмма. Если последовательность или параллельность выполнения действий окончательно не ясна, эксперты могут быть опрошены вторично (возможно, с использованием черновых вариантов незаконченных диаграмм) для получения недостающей информации. Важно, однако, различать предполагаемую (появляющуюся из- за недостатка информации о связях) и явную (ясно указанную в описании эксперта) параллельности.

Итак, IDEF3 – это способ описания бизнес-процессов, который нужен для описания положения вещей как упорядоченной последовательности событий с одновременным описанием объектов, имеющих непосредственное отношение к процессу. IDEF3 хорошо приспособлен для сбора данных, требующихся для проведения структурного анализа системы. Кроме того, IDEF3 применяется при проведении стоимостного анализа поведения моделируемой системы.

2.4.2 Методология функционального моделирования IDEF0

Методология функционального моделирования IDEF0 – это технология описания системы в целом как множества взаимозависимых действий, или функций. Важно отметить функциональную направленность IDEF0 – функции системы исследуются независимо от объектов, которые обеспечивают их выполнение. «Функциональная» точка зрения позволяет четко отделить аспекты назначения системы от аспектов ее физической реализации. На рисунке 2.17 приведен пример типовой диаграммы IDEF0.

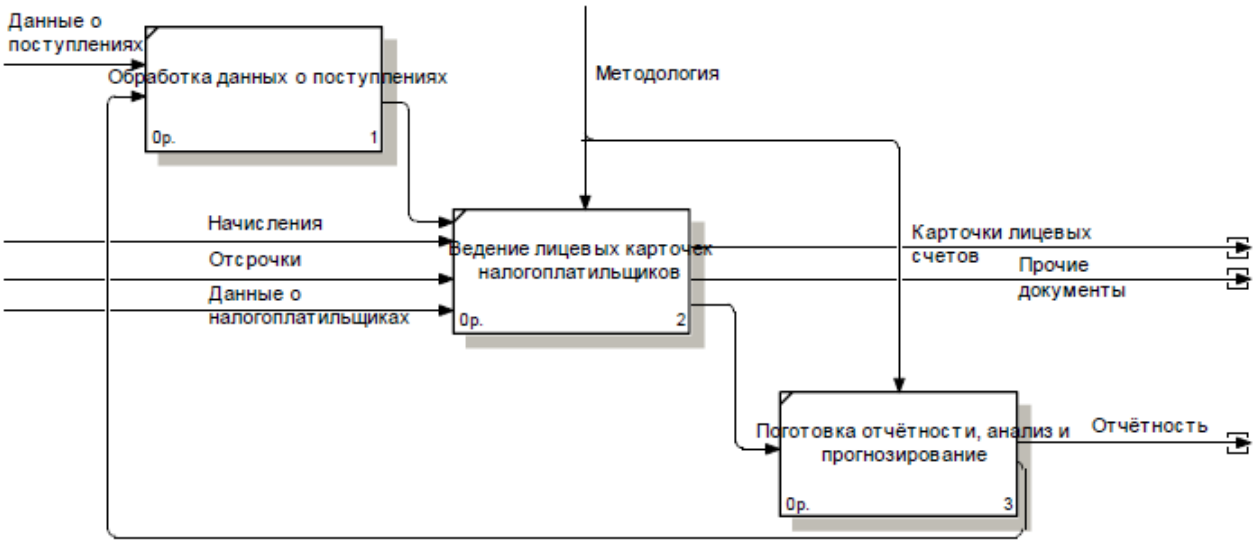


Рисунок 2.17 - Пример диаграммы IDEF0

Наиболее часто IDEF0 применяется как технология исследования и проектирования систем на логическом уровне. По этой причине он, как правило, используется на ранних этапах разработки проекта, до IDEF3-моделирования для сбора данных и моделирования процесса «как есть». Результаты IDEF0-анализа могут применяться при проведении проектирования с использованием моделей IDEF3 и диаграмм потоков данных.

2.4.2.1 Синтаксис и семантика моделей IDEF0

Модели IDEF0. IDEF0 сочетает в себе небольшую по объему графическую нотацию (она содержит только два обозначения: блоки и стрелки) со строгими и четко определенными рекомендациями, в совокупности предназначенными для построения качественной и понятной модели системы.

Методология IDEF0 в некоторой степени напоминает рекомендации, существующие в книгоиздательском деле, часто набор напечатанных моделей IDEF0 организуется в брошюру (называемую в терминах IDEF0 комплект), имеющую содержание, глоссарий и другие элементы, характерные для законченной книги.

Первый шаг при построении модели IDEF0 заключается в определении назначения модели – набора вопросов, на которые должна отвечать модель.

Набор вопросов можно сравнить с предисловием, в котором раскрывается назначение книги.

Границы моделирования предназначены для обозначения ширины охвата предметной области и глубины детализации и являются логическим продолжением уже определенного назначения модели. Как читающий модель, так и непосредственно ее автор должны понимать степень детальности ответов на поставленные в назначении модели вопросы.

Следующим шагом указывается предполагаемая целевая аудитория, для нужд которой создается модель. Зачастую от выбора целевой аудитории зависит уровень детализации, с которым должна создаваться модель. Перед построением модели необходимо иметь представление о том, какие сведения о предмете моделирования уже известны, какие дополнительные материалы и (или) техническая документация для понимания модели могут быть необходимы целевой аудитории, какие язык и стиль изложения являются наиболее подходящими.

Под точкой зрения понимается перспектива, с которой наблюдалась система при построении модели. Точка зрения выбирается таким образом, чтобы учесть уже обозначенные границы моделирования и назначение модели. Однажды выбранная точка зрения остается неизменной для всех элементов модели. При необходимости могут быть созданы другие модели, отображающие систему с других точек зрения. Вот несколько примеров точек зрения при построении моделей: клиент, поставщик, владелец, редактор.

Действия. Действие, обычно в IDEF0 называемое функцией, обрабатывает или переводит входные параметры (сырье, информацию и т. п.) в выходные. Поскольку модели IDEF0 представляют систему как множество иерархических (вложенных) функций, в первую очередь должна быть определена функция, описывающая систему в целом – контекстная функция. Функции изображаются на диаграммах как поименованные прямоугольники, или функциональные блоки. Имена функций в IDEF0 подбираются по сходным правилам с именами действий в IDEF3 – с использованием глаголов или отглагольных существительных. Важно подбирать имена таким образом, чтобы они отражали систему так, как если бы она обозревалась с точки зрения, выбранной для моделирования.

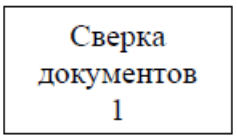


Рисунок 2.18 - Функциональный блок IDEF0

Пример функционального блока приведен на рисунке 2.18.

Выше мы определяли IDEF0-модели как иерархическое множество вложенных блоков. Любой блок может быть декомпозирован на составляющие его блоки. Декомпозицию часто ассоциируют с моделированием «сверху вниз», однако это не совсем верно. Функциональную декомпозицию корректнее определять как моделирование «снаружи вовнутрь», в котором мы рассматриваем систему наподобие луковицы, с которой последовательно снимаются слои.

Границы и связи. Чтобы быть полезным, описание любого блока должно, как минимум, включать в себя описание объектов, которые блок создает в результате своей работы («выхода»), и объектов, которые блок потребляет или преобразует («вход»).

В IDEF0 также моделируются управление и механизмы исполнения. Под управлением понимаются объекты, воздействующие на способ, которым блок преобразует вход в выход. Механизм исполнения – объекты, которые непосредственно выполняют преобразование входа в выход, но не потребляются при этом сами по себе.

Для отображения категорий информации, присутствующих на диаграммах IDEF0, существует аббревиатура ICOM, отображающая четыре возможных типа стрелок:

I (Input) – вход – нечто, что потребляется в ходе выполнения процесса;

C (Control) – управление – ограничения и инструкции, влияющие на ход выполнения процесса;

O (Output) – выход – нечто, являющееся результатом выполнения процесса;

M (Mechanism) – исполняющий механизм – нечто, что используется для выполнения процесса, но не потребляется само по себе. Рисунок 2.19 показывает четыре возможных типа стрелок в IDEF0, каждый из типов соединяется со своей стороной функционального блока.

Для названия стрелок, как правило, употребляются имена существительные. Стрелки

могут представлять собой людей, места, вещи, идеи или события. Как и в случае с функциональными блоками, присвоение имен всем стрелкам на диаграмме является только необходимым условием для понимания читателем сути изображенного. Отдельное описание каждой стрелки в текстовом виде может оказаться критическим фактором для построения точной и полезной модели.

Рисунок 2.19 - Соединение стрелок со сторонами функционального блока

Стрелки входа. Вход представляет собой сырье, или информацию, потребляемую или преобразуемую функциональным блоком для производства выхода. Стрелки входа всегда направлены в левую сторону прямоугольника, обозначающего в IDEF0 функциональный блок. Наличие входных стрелок на диаграмме не является обязательным, так как возможно, что некоторые блоки ничего не преобразуют и не изменяют. Примером блока, не имеющего входа, может служить блок «Принятие решения руководством», где для принятия решения анализируется несколько факторов, но ни один из них непосредственно не преобразуется и не потребляется в результате принятия какого-либо решения.

Стрелки управления. Стрелки управления отвечают за регулирование того, как и когда выполняется функциональный блок, и, если он выполняется, какой выход получается в результате его выполнения. Так как управление контролирует поведение функционального блока для обеспечения создания желаемого выхода, каждый функциональный блок должен иметь, как минимум, одну стрелку управления. Стрелки управления всегда входят в функциональный блок сверху.

Управление часто существует в виде правил, инструкций, законов, политики, набора необходимых процедур или стандартов. Влияя на работу блока, оно непосредственно не потребляется и не трансформируется в результате. Может оказаться, что целью функционального блока является как раз изменение того или иного правила, инструкции, стандарта и т. п. В этом случае стрелка, содержащая соответствующую информацию, должна рассматриваться не как управление, а как вход функционального блока.

Управление можно рассматривать как специфический вид входа. В случаях, когда неясно, относить ли стрелку к входу или к управлению, предпочтительно относить ее к управлению до момента, пока неясность не будет разрешена.

Стрелки выхода. Выход – это продукция или информация, получаемая в результате работы функционального блока. Каждый блок должен иметь, как минимум, один выход. Действие, которое не производит никакого четко определяемого выхода, не должно моделироваться вообще (по меньшей мере, должно рассматриваться в качестве одного из первых кандидатов на исключение из модели).

При моделировании непроизводственных предметных областей выходами, как правило, являются данные, в каком-либо виде обрабатываемые функциональным блоком. В этом случае важно, чтобы названия стрелок входа и выхода были достаточно различимы по своему смыслу. Например, блок «Прием пациентов» может иметь стрелку «Данные о пациенте» как на входе, так и на выходе. В такой ситуации входящую стрелку можно назвать «Предварительные данные о пациенте», а исходящую – «Подтвержденные данные о пациенте».

Стрелки механизма исполнения. Механизмы являются ресурсом, который непосредственно исполняет моделируемое действие. С помощью механизмов исполнения могут моделироваться: ключевой персонал, техника и (или) оборудование. Стрелки механизма исполнения могут отсутствовать в случае, если оказывается, что они не являются

необходимыми для достижения поставленной цели моделирования.

Комбинированные стрелки. В IDEF0 существует пять основных видов комбинированных стрелок: «Выход-вход», «Выход – управление», «Выход – механизм исполнения», «Выход – обратная связь на управление» и «Выход – обратная связь на вход».

Стрелка «Выход-вход» применяется, когда один из блоков должен полностью завершить работу перед началом работы другого блока. Так, на рисунке 2.20 формирование счета должно предшествовать приему заказа.

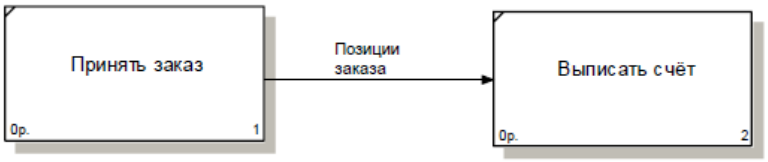


Рисунок 2.20 - Комбинация стрелок «Выход-вход»

Стрелка «Выход – управление» отражает ситуацию преобладания одно- го блока над другим, когда один блок управляет работой другого. На рисунке 2.21 принципы формирования инвестиционного портфеля управляют поведением брокеров на бирже.

Рисунок 2.21 - Комбинированная стрелка «Выход – управление»

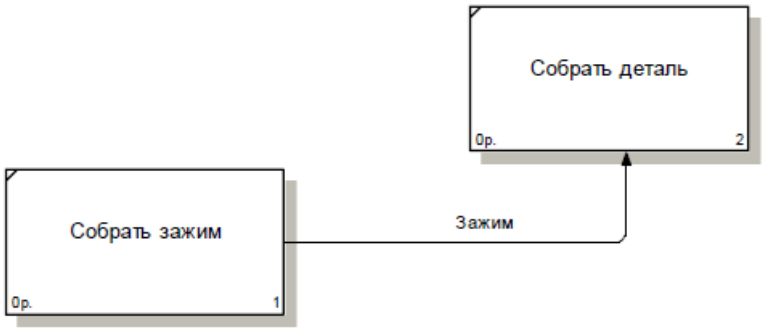


Рисунок 2.22 - Комбинированная стрелка «Выход – механизм исполнения»

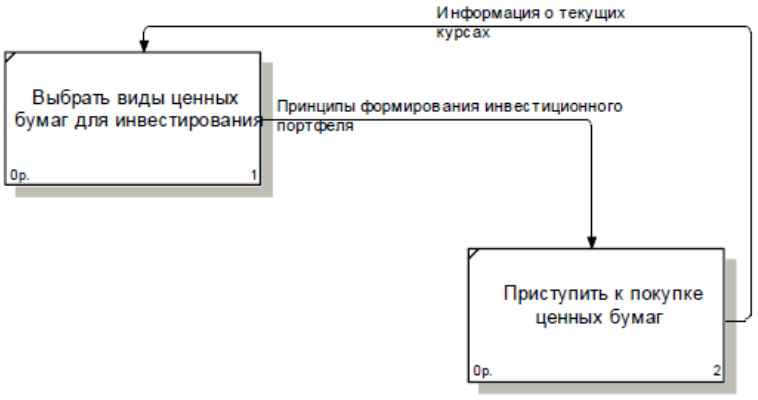


Рисунок 2.23 - Комбинированная стрелка «Выход – обратная связь на управление»

Стрелки «Выход – механизм исполнения» встречаются реже и отражают ситуацию, когда выход одного функционального блока применяется в качестве оборудования для работы другого

блока. На рисунке 2.22 зажим, устройство, используемое для закрепления детали во время ее сборки, должно быть собрано для того, чтобы выполнить сборку детали.

Обратные связи на вход и на управление применяются в случаях, когда зависимые блоки формируют обратные связи для управляющих ими блоков.

На рисунке 2.23 получаемая от брокеров информация о текущих биржевых курсах применяется для корректировки стратегии игры на бирже.

Стрелка «Выход – обратная связь на вход» обычно применяется для описания циклов повторной обработки чего-либо. Рисунок 2.24 может служить примером применения стрелки такого типа. Кроме того, связи «Выход – обратная связь на вход» могут применяться в случае, если бракованная продукция может заново использоваться в качестве сырья, как это происходит, например, при производстве оконного стекла, когда разбитое в процессе производства стекло перемалывается и переплавляется заново вместе с обыкновенным сырьем.

Разбиение и соединение стрелок. Выход функционального блока может использоваться в нескольких других блоках. Фактически чуть ли не главная ценность IDEF0 заключается в том, что эта методология помогает выявить взаимозависимости между блоками системы. Соответственно IDEF0 предусматривает как разбиение, так и соединение стрелок на диаграмме. Разбитые на несколько частей стрелки могут иметь наименования, отличающиеся от наименования исходной стрелки. Исходная и разбитые (или объединенные) стрелки в совокупности называются связанными. Такая техника обычно применяется для того, чтобы отразить использование в процессе только части сырья или информации, обозначаемых исходной стрелкой (рисунок 2.25). Аналогичный подход применяется и к объединяемым стрелкам.

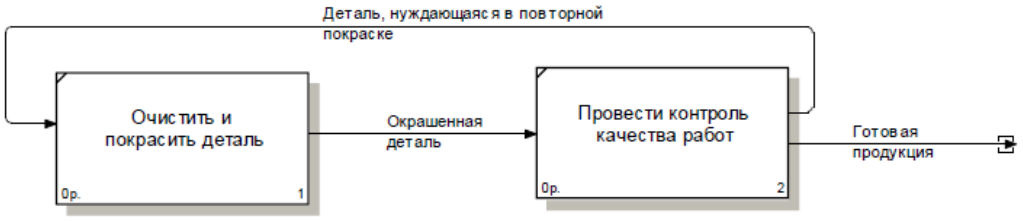


Рисунок 2.24 - Комбинированная стрелка «Выход – обратная связь на вход»

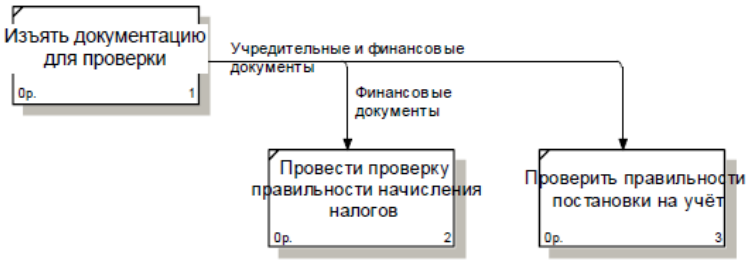


Рисунок 2.25 - Разбитая на две части и переименованная стрелка

Туннели. Понятие «связанные стрелки» используется для управления уровнем детализации диаграмм. Если одна из стрелок диаграммы отсутствует на родительской диаграмме (например, ввиду своей несущественности для родительского уровня) и не связана с другими стрелками той же диаграммы, точка входа этой стрелки на диаграмму или выхода с нее обозначается туннелем. На рисунок 2.26 , например, стрелка «Корпоративная информационная система» – важный механизм исполнения для данной диаграммы, но, возможно, она более нигде не используется в модели. Туннель в данном случае используется как альтернатива загромождению родительских диаграмм помещением на них несущественных для их уровня стрелок.

Рисунок 2.26 - Пример применения туннеля

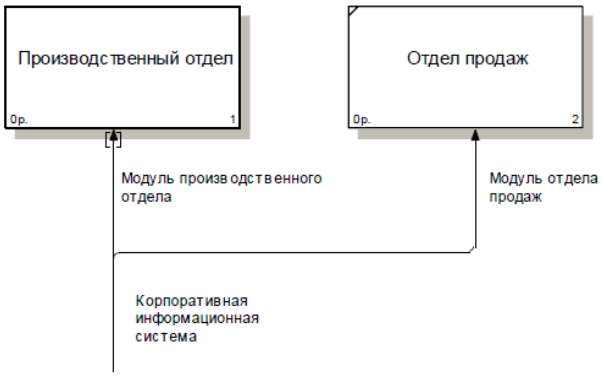


Рисунок 2.27- Еще один пример применения туннеля

Кроме того, туннели применяются для отражения ситуации, когда стрелка, присутствующая на родительской диаграмме, отсутствует в диаграмме декомпозиции соответствующего блока. На рисунке 2.27 туннель у стрелки «Модуль производственного отдела» обозначает, что на диаграмме декомпозиции производственного отдела отсутствует стрелка механизма управления с соответствующим наименованием.

2.4.2.2 Построение моделей IDEF0

Здесь мы рассмотрим методику построения моделей IDEF0 более подробно. Диаграммы. На рисунке 2.28 типовая диаграмма IDEF0 показана вместе с находящейся на ее полях служебной информацией. Служебная информация состоит из хорошо выделенных верхнего и нижнего колонтитулов (заголовка и «подвала»). Элементы заголовка используются для отслеживания процесса создания модели. Элементы «подвала» отображают наименование модели, к которой относится диаграмма, и показывают ее расположение относительно других диаграмм модели.

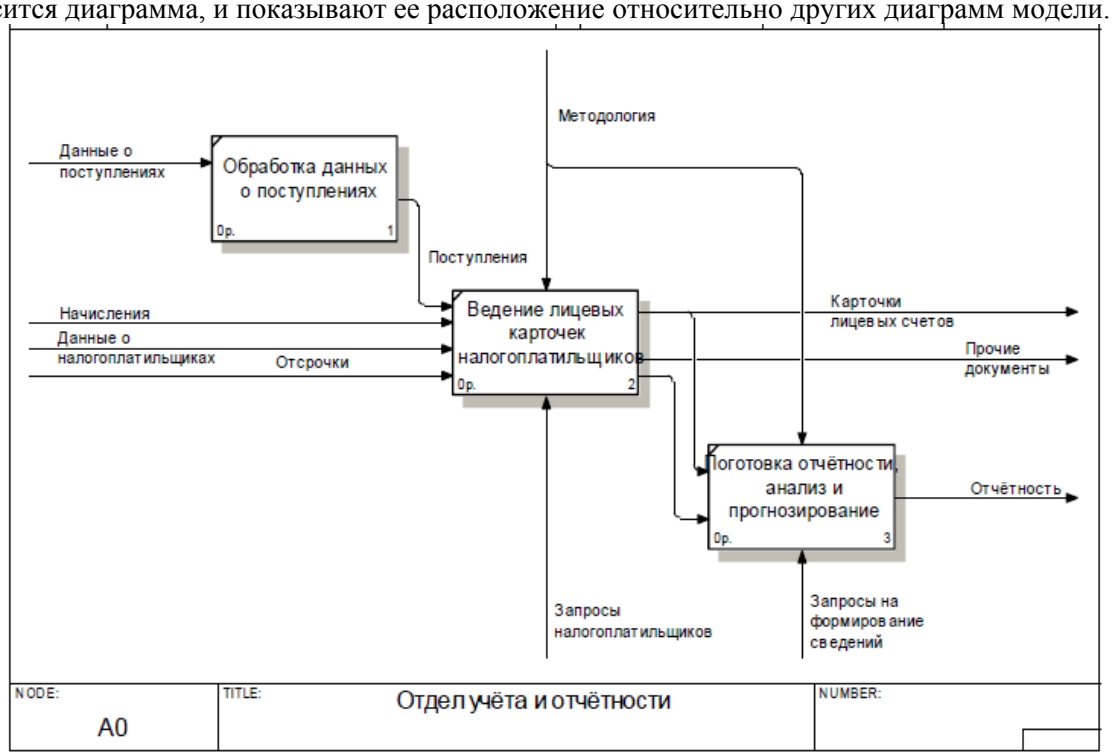


Рисунок 2.28 - IDEF0-диаграмма со служебной информацией на полях

Все элементы заголовка диаграммы перечислены в таблице 2.12.

Таблица 2.12 - Элементы заголовка диаграммы IDEF0

Поле	Назначение
USED AT	Используется для отражения внешних ссылок на данную диаграмму (заполняется на печатном документе вручную)
Author, date, project	Содержит Ф. И. О. автора диаграммы, дату создания, дату последнего внесения изменений, наименование проекта, в рамках которого она создавалась
Notes 1 ... 10	При ручном редактировании диаграмм пользователи могут зачеркивать цифру каждый раз, когда они вносят очередное исправление
Status	Статус отражает состояние разработки или утверждения данной диаграммы. Это поле используется для реализации формального процесса публикации с шагами пересмотра и утверждения
Working	Новая диаграмма, глобальные изменения или новый автор для существующей диаграммы
Draft	Диаграмма достигла некоторого приемлемого для читателей уровня и готова для представления на утверждение
Recommended	Диаграмма одобрена и утверждена, какие-либо изменения не предвидятся
Publication	Диаграмма готова для окончательной печати и публикации
Reader	Ф. И. О. читателя
Date	Дата знакомства читателя с диаграммой
Context	Набросок расположения функциональных блоков на родительской диаграмме, на котором подсвечен декомпозируемый данной диаграммой блок. Для диаграммы самого верхнего уровня (контекстной диаграммы) в поле помещается контекст TOP

Все элементы нижней строки диаграммы перечислены в таблице 2.13.

Таблица 2.13 - Элементы нижней строки диаграммы IDEF0

Поле	Назначение
Node	Номер диаграммы, совпадающий с номером родительского функционального блока
Title	Имя родительского функционального блока
Number	Уникальный идентификатор данной версии данной диаграммы. Таким образом, каждая новая версия данной диаграммы будет иметь новое значение в этом поле. Как правило, C-Number состоит из инициалов автора (которые предполагаются уникальными среди всех аналитиков проекта) и последовательного уникального идентификатора, например SDO005. При публикации эти номера могут быть заменены стандартными номерами страниц. Если диаграмма замещает другую диаграмму, номер заменяемой диаграммы может быть заключен в скобки – SDO005 (SDO004). Это обеспечивает хранение истории изменений всех диаграмм модели

Цикл «эксперт – аналитик». Подобно циклу «автор – редактор», применяющемуся в книгоиздательском деле, диаграммы IDEF0 пересматриваются и изменяются для

обеспечения точности отражения предметной области и улучшения своего качества.

Для каждого рецензента автором, как правило, подготавливается свой набор диаграмм. Предложения по изменениям и исправлениям возвращаются рецензентами автору для внесения их в модель. При возникновении разногласий между автором и рецензентом спорная диаграмма обычно рассылается всем рецензентам для достижения группового консенсуса.

Формально механизм рецензирования и модификации диаграмм поддерживается полями Status и нумерацией диаграмм, контроль истории изменений – полем Field (см. таблицу 2.12).

Построение моделей. Ни одна модель не должна строиться без ясного осознания объекта и целей моделирования. Выбранное определение цели моделирования должно отвечать на следующие вопросы:

Почему моделируется данный процесс?

Что выявит данная модель?

Как ознакомившиеся с этой моделью смогут ее применить?

Следующее предложение может служить примером формулирования цели моделирования. Выявить задачи каждого работника компании и понять в целом взаимосвязь между отдельно взятыми задачами для разработки руководства по обучению новых сотрудников.

Модели строятся для того, чтобы ответить на набор поставленных вопросов. Такие вопросы формулируются на ранних стадиях моделирования и впоследствии служат основой для четкого и краткого определения цели моделирования. Примерами таких вопросов могут быть:

Каковы задачи менеджера?

Каковы задачи клерка?

Кто контролирует работу?

Какая технология нужна для выполнения каждого шага? и т. п.

Точка зрения. С методической точки зрения при моделировании полезно использовать мнение экспертов, имеющих разные взгляды на предметную область, однако каждая отдельно взятая модель должна разрабатываться исходя из единственной заранее определенной точки зрения. Часто другие точки зрения вкратце документируются в прикрепленных диаграммах FEO (см. ниже) исключительно для наглядности изложения.

Точку зрения нужно подбирать достаточно аккуратно, основой для выбора должна служить поставленная цель моделирования. Наименованием точки зрения может быть наименование должности, подразделения или роли (например, руководитель отдела или менеджер по продажам). Как и в случае с определением цели моделирования, четкое определение точки зрения необходимо для обеспечения внутренней целостности модели и предотвращения постоянного изменения ее структуры. Может оказаться необходимым построение моделей с разных точек зрения для детального отражения всех особенностей, выделенных в системе функциональных блоков.

Границы моделирования. Одним из положительных результатов построения функциональных моделей оказывается прояснение границ моделирования системы, а в целом и ее основных компонентов. Хотя и предполагается, что в процессе работы над моделью будет происходить некоторое изменение границ моделирования, их вербальное (словесное) описание должно поддерживаться с самого начала для обеспечения координации работы участвующих в проекте аналитиков. Как и при определении цели моделирования, отсутствие границ затрудняет оценку степени завершенности модели, поскольку границы моделирования имеют тенденцию к расширению с ростом размеров модели.

Границы моделирования имеют два компонента: ширину охвата и глубину детализации. Ширина охвата обозначает внешние границы моделируемой системы. Глубина детализации определяет степень подробности, с которой нужно проводить декомпозицию функциональных блоков.

Чтобы облегчить правильное определение границ моделирования при разработке моделей IDEF0, существенные усилия затрачиваются на разработку и рецензирование контекстной диаграммы IDEF0 (диаграммы «самого верхнего» уровня). Иногда даже прибегают к построению дополнительной диаграммы для отображения уровня, более высокого, чем контекстный, для данной модели, что позволяет обозначить систему, внутри которой располагается объект для моделирования. Существенные затраты на разработку контекстной диаграммы вполне оправданы, поскольку она является своего рода «точкой отсчета» для остальных диаграмм модели и вносимые в нее изменения каскадом отражаются на всех лежащих

ниже уровнях.

Когда границы моделирования понятны, становятся ясными и причины, по которым некоторые объекты системы не вошли в модель.

Выбор наименования контекстного блока. Рекомендуемой последовательностью действий при построении модели с нуля являются: формулирование цели моделирования, выбор точки зрения, определение границ моделирования. Наименование контекстного блока – функционального блока самого высокого уровня – обобщает определение границ моделирования.

Правила подбора имени для контекстного блока в целом не отличаются от общих правил наименования функциональных блоков, поэтому для них обычно подбирают обобщающие названия, типа «Управление отделом по работе с клиентами», «Обработка заказов» и т. п.

Определение стрелок на контекстной диаграмме. Стрелки диаграмм IDEF0 обычно проще проектировать в следующем порядке: выход, вход, механизм исполнения, управление. Каждый функциональный блок обозначает отдельную функцию, и эта функция часто имеет ясно и кратко описываемые результаты работы. Наличие неясностей при анализе выходов того или иного функционального блока – возможный сигнал необходимости проведения ре- инжиниринга рассматриваемого бизнес-процесса.

Определение выходов. После идентификации возможных выходов полезно провести анализ модели на предмет покрытия ею всех возможных сценариев поведения процесса. Это означает, что если существует вероятность возникновения той или иной ситуации в ходе процесса, модель отражает возможность возникновения такой ситуации. Многие начинающие аналитики забывают отразить негативные результаты работы функциональных блоков. Например, блок «Провести экзамен по вождению» определенно произведет поток водителей, только что получивших права, но вполне правомерно ожидать и потока лиц, не сдавших экзамен. Негативные результаты часто используются в качестве обратных связей, анализ на их наличие должен проводиться для каждого блока. Важным также является необходимость включения в модель спорных стрелок, принятие решения о наличии которых в модели вполне можно переложить на плечи рецензирующих модель экспертов.

Определение входов. Входы можно рассматривать как особым образом преобразуемые функциональными блоками для производства выхода сырье или информацию. В производственных отраслях определить, как входное сырье преобразуется в готовую продукцию, обычно довольно просто. Однако при моделировании информационных потоков входной поток данных может представляться не потребляемым и не обрабатываемым вообще. Случаи, когда входящие и исходящие стрелки называются в точности одинаково, крайне редки и в основном указывают на бесполезность данного блока для системы в целом или на некорректный выбор имени для исходящей стрелки. Решением может служить применение более подробного описания для входящих и исходящих потоков данных. Например, вход может иметь название «Предварительный диагноз пациента», а выход – «Уточненный диагноз пациента».

Определение механизмов исполнения. После создания входов и выходов можно приступить к рассмотрению механизмов исполнения, или ресурсов, относящихся к функциональному блоку. В понятие механизма исполнения входят персонал, оборудование, информационные системы и т. п. Например, функциональный блок «Собрать деталь» может потребовать использования какого-либо оборудования, например гаечного ключа. При приеме экзаменов на водительские права механизмом исполнения является инспектор ГИБДД. Как правило, определить механизмы исполнения для функциональных блоков довольно просто.

Определение управления. Должно быть определено управление, контролирующее ход работы функционального блока. Все функциональные блоки в IDEF0 должны иметь хотя бы одно управление. В случаях, когда не ясно, относить ли стрелку к входу или к управлению, следует ее рисовать как управление. Важно помнить, что управление можно рассматривать как особую форму входа функционального блока.

Когда контекстная диаграмма представляется завершенной, попробуйте задать следующие вопросы:

Обобщает ли диаграмма моделируемый бизнес-процесс?

Согласуется ли диаграмма с границами моделирования, точкой зрения и целью моделирования?

Подходит ли выбранный уровень детализации стрелок для контекстного блока? (Обычно на контекстной диаграмме рекомендуется рисовать не более шести стрелок каждого типа.)

Нумерация блоков и диаграмм. Все функциональные блоки IDEF0 нумеруются. В номерах допускается использование префиксов произвольной длины, но в подавляющем большинстве моделей используется префикс А. Номер блока проставляется за префиксом. Контекстный блок всегда имеет номер А0.

Префикс повторяется для каждого блока модели. Номера используются для отражения уровня декомпозиции, на котором находится блок. Блок А0 декомпозируется в блоки А1, А2, А3 и т. д. А1 декомпозируется в А11, А12, А13 и т. д. А11 декомпозируется в А111, А112, А113 и т. д. Для каждого уровня декомпозиции в конец номера добавляется одна цифра.

Связь между диаграммой и ее родительским функциональным блоком. Функциональный блок декомпозируется, если необходимо детально описать его работу. При декомпозиции блока полезно рассмотреть его жизненный цикл, это поможет определить функциональные блоки получающейся детской диаграммы. Например, жизненный цикл 1 блока «Поджарить бифштекс» может выглядеть как следующая последовательность: подготовить продукты – отбить мясо – разогреть масло и т. д.

При моделировании IDEF0 важно иметь в виду, что граница детской диаграммы есть граница родительского функционального блока. Это означает, что вся работа выполняется блоками самого нижнего уровня. В отличие от иерархии, применяемой в структурном программировании, блоки верхнего уровня не являются субъектами управления для блоков нижнего уровня. Это означает, что в IDEF0 дети – это те же объекты, что и их родители, только показанные с большей детализацией. Действия генерального директора компании на диаграммах IDEF0 могут отражаться рядом с действиями простых рабочих.

На концах граничных стрелок (начинающихся или заканчивающихся за пределами диаграммы) детских диаграмм помещаются коды ICOM, чтобы показать, где находится соответствующая стрелка на родительской диаграмме (рисунок 2.29). Они нужны для проверки целостности модели и могут быть полезны, когда порядок расположения стрелок на детской диаграмме отличается от порядка их расположения на родительской диаграмме. Код ICOM состоит из латинской буквы I, C, O или M и числа, показывающего расположение стрелки на родительской диаграмме в порядке сверху вниз или слева направо.

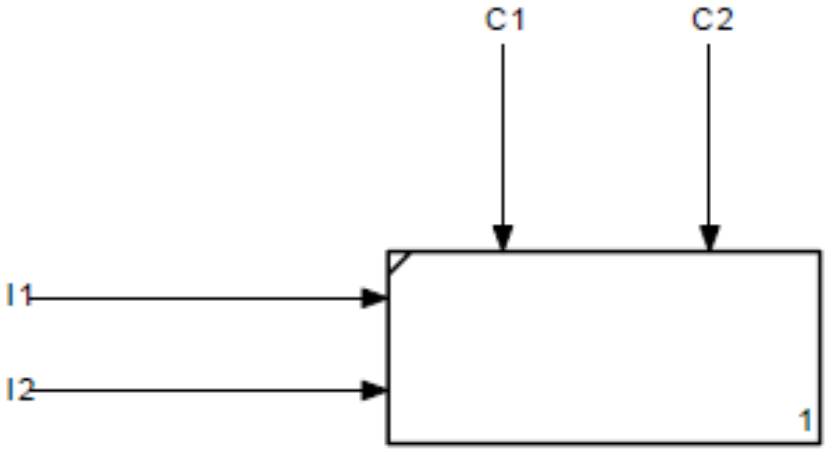


Рисунок 2.29 - ICOM-коды на граничных стрелках

Два подхода к началу моделирования («в ширину» и «в глубину»). Модели могут проектироваться с использованием подхода «в ширину», когда каждая диаграмма максимально детализируется перед своей декомпозицией, и с подходом «в глубину», когда сначала определяется иерархия блоков, а затем создаются соединяющие их стрелки. Естественно, возможно применение комбинации этих подходов, причем иерархия блоков может иногда немного измениться после того, как нарисованы стрелки. Это происходит из-за того, что создание стрелок может изменить понимание.

План конспекта:

1. Введение в системный анализ.
2. Анализ проблемы и моделирование предметной области с использованием системного подхода.
3. Методология ARIS. Стандарты IDEF0 – IDEF3.

Работа с литературой:

Рекомендуемые источники информации (№ источника)			
Основная	Дополнительная	Методическая	Интернет-ресурсы
1-3	1-3	1-3	1-2

Тема самостоятельного изучения № 3
Анализ требований и их формализация.

Вид деятельности студентов: самостоятельное изучение литературы
Итоговый продукт самостоятельной работы: конспект
Средства и технологии оценки: собеседование
Краткие теоретические сведения

3 АНАЛИЗ ТРЕБОВАНИЙ И ИХ ФОРМАЛИЗАЦИЯ

3.1 Методы определения требований

3.1.1 Интервьюирование

Одним из наиболее важных и понятных методов получения требований является *интервью с клиентом;* это метод, который можно использовать практически в любой ситуации.

Одна из основных задач интервьюирования – сделать все возможное, чтобы предубеждения и предпочтения интервьюируемых не повлияли на свободный обмен информацией. Это сложная проблема. Социология учит нас, что невозможно воспринимать окружающий мир, не фильтруя его в соответствии со своим происхождением и накопленным опытом.

3.1.1.1 Этапы проведения интервью

Процесс проведения интервью состоит из следующих этапов:

Разработка вопросов (образец вопросов интервью представлен в таблице 3.1).

Выбор опрашиваемых пользователей. Существует несколько групп пользователей: персонал начального уровня, организаторы проекта среднего уровня, менеджеры и другие заказчики особого рода – генеральные директоры и вице-президенты, научные сотрудники, обычные пользователи системы.

Планирование контактов.

Проведение интервью. Интервью можно провести по телефону, персонально или с помощью Интернета (видеоконференция, чат, электронная почта), однако лучшим способом осуществления интервью является непосредственное общение, лицом к лицу.

Завершение встречи и определение последующих действий.

Таблица 3.1 - Обобщенное практически контекстно-свободное интервью

Часть I. Определение профиля заказчика или пользователя
Имя
Компания
Отрасль
Должность
(Вышеприведенная информация, как правило, может быть внесена заранее.)
Каковы ваши основные обязанности?
Что вы в основном производите?
Для кого?
Как измеряется успех вашей деятельности?
Какие проблемы влияют на успешность вашей деятельности?
Какие тенденции, если такие существуют, делают вашу работу проще или сложнее?
Часть II. Оценка проблемы
Для каких проблем (прикладного типа) вы ощущаете нехватку хороших решений?
Назовите их. (Замечание. Не забывайте спрашивать: «А еще?».)
Для каждой проблемы выясняйте следующее.
Почему существует эта проблема?
Как она решается в настоящее время?
Как заказчик (пользователь) хотел бы ее решать?
Часть III. Понимание пользовательской среды
Кто такие пользователи?
Какое у них образование?
Каковы их навыки в компьютерной области?
Имеют ли пользователи опыт работы с данным типом приложений?
Какая платформа используется?
Каковы ваши планы относительно будущих платформ?
Используются ли дополнительные приложения, которые имеют отношение к данному приложению? Если да, то пусть о них немного расскажут.
Каковы ожидания заказчика относительно практичности продукта?
Сколько времени необходимо для обучения?

В каком виде должна быть представлена справочная информация для пользователя (в интерактивном или в виде печатной копии)?

Часть IV. Резюме (перечисляются основные пункты, чтобы проверить, все ли правильно вы поняли)

Итак, вы сказали мне (перечислите описанные заказчиком проблемы своими словами)

Адекватно ли этот список представляет проблемы, которые имеются при существующем решении?

Какие еще проблемы (если такие существуют) вы испытываете?

Часть V. Предположения, аналитика относительно проблемы заказчика (проверенные или непроверенные предположения)

(те проблемы, которые не были упомянуты)

Какие проблемы, если они есть, связаны с (перечислите все потребности или дополнительные проблемы, которые, по-вашему, может испытывать заказчик или пользователь)

Для каждой из указанных проблем выясните следующее.

Является ли она реальной?

Каковы ее причины?

Как она решается в настоящее время?

Как бы заказчик (пользователь) хотел ее решать?

Насколько важно для заказчика (пользователя) решение этой проблемы в сравнении с другими, упомянутыми им?

Часть VI. Оценка предлагаемого вами решения (если это уместно)

(Охарактеризуйте основные возможности предлагаемого вами решения.
А потом задайте пользователю следующие вопросы.)

Что, если вы сможете?

Как вы расцениваете важность этого?

Часть VII. Оценка возможности

Кто в организации нуждается в данном приложении?

Сколько пользователей указанных типов будет использовать его?

Насколько значимо для вас успешное решение?

Часть VIII. Оценка необходимого уровня надежности и производительности, а также потребности в сопровождении

Каковы ваши ожидания относительно надежности?

Какой, по-вашему, должна быть производительность?

Будете ли вы заниматься поддержкой продукта или этим будут заниматься другие?

Испытываете ли вы потребности в поддержке?

Что вы думаете о доступе для сопровождения и обслуживания?

Каковы требования относительно безопасности?

Какие требования относительно установки и конфигурации?

Существуют ли специальные требования по лицензированию?

Как будет распределено программное обеспечение?

Есть ли требования на маркировку и упаковку?

Часть IX. Другие требования

Существуют ли законодательные требования, требования информационной среды, инструкции или другие стандарты, которых необходимо придерживаться? Нет ли других требований, о которых нам следовало бы знать?

Часть X. Окончание

Существуют ли другие вопросы, которые мне следовало бы вам задать?

Если мне еще понадобится задать вам несколько вопросов, могу ли я вам позвонить?

Будете ли вы принимать участие в обсуждении требований?

Часть XI. Заключение аналитика

После интервью, пока его данные еще свежи в вашей памяти, зафиксируйте три потребности или проблемы с наивысшими приоритетами, выявленные вами в беседе с данным заказчиком (пользователем)

3.1.2 «Мозговой штурм» и отбор идей

«Мозговой штурм» – это метод проведения собрания, при котором группа людей пытается найти решение специфической проблемы посредством накопления всех спонтанно возникающих идей.

Данный процесс имеет ряд очевидных преимуществ. Поддерживает участие всех присутствующих. Позволяет участникам развивать идеи друг друга.

Ведущий или секретарь ведет запись всего хода обсуждения.

Его можно применять при различных обстоятельствах.

Как правило, в результате получаем множество возможных решений для любой поставленной проблемы.

Метод способствует свободному мышлению, не ограниченному обычными рамками.

«Мозговой штурм» состоит из двух фаз: генерация идей и их отбор.

Основная цель на этапе генерации состоит в том, чтобы описать как можно больше идей, не обязательно глубоких. На этапе отбора главной задачей является анализ всех возникших идей. Отбор идей включает в себя отсечение, организацию, упорядочение, развитие, группировку, уточнение и т. п.

3.1.2.1 Генерация идей

«Мозговой штурм» можно производить различными способами. Ниже описывается один из простых процессов. Все основные участники собираются в одной комнате, и им раздаются материалы для заметок. Это может быть просто стопка бумаги и черный толстый маркер. Листы бумаги должны быть не менее 7×12 см и не более 12×17 см. Каждому участнику нужно выдать не менее 25 листов на каждый сеанс «мозгового штурма».

Каждый участник сеанса «мозгового штурма» исполняет одну из трех ролей: лидер, секретарь или член команды. Лидер отвечает за направление процесса в правильное русло, за порядок и помогает секретарю делать запи- си. Секретарь записывает все идеи таким образом, чтобы каждый человек, находящийся в комнате, мог видеть эти записи. Участники генерируют идеи.

Ведущий объясняет правила проведения «мозгового штурма» (рисунок 3.1) и определяется цель заседания.

Правила проведения «мозгового штурма»
1. Не допускается критика или дебаты.
2. Дайте свободу фантазии.
3. Генерируйте как можно больше идей.
4. Переделывайте и комбинируйте идеи.

Рисунок 3.1- Правила проведения «мозгового штурма»

По мере создания идей секретарь просто собирает их и прикрепляет к стене комнаты заседаний. Большинство заседаний по генерации идей длится около часа (иногда 2–3 часа).

3.1.2.2 Отбор идей

После завершения фазы генерации идей начинается процесс отбора, который состоит из нескольких этапов.

1. Отсечение. Заключается в отсечении тех идей, которые не достойны внимания. Если обнаруживается две одинаковые идеи, эти идеи объединяются.

2. Группировка идей. Группы получают названия в зависимости от того, по какому принципу осуществляется группировка. Например, группы могут иметь следующие названия: новые функции, вопросы производительности, предложения по усовершенствованию существующих функций, интерфейс пользователя и вопросы простоты обращения и т. д. Для любой из этих групп можно возобновить генерацию идей, если окажется, что процесс группировки стимулировал возникновение новых идей или некоторая область важных функциональных возможностей осталась неохваченной.

3. Определение функций. Ведущий перечисляет все оставшиеся идеи и просит авторов дать их описание, состоящее из одного предложения (таблица 3.2).

4. Расстановка приоритетов.

Таблица 3.2 - Пример определения функций

Область применения приложения	Штурмуемая функция	Описание функции
Автоматизация домашнего освещения	Автоматическое задание освещения	Домовладелец может предварительно задавать основанные на времени последовательности возникновения определенных осветительных событий в зависимости от времени дня
Система ввода заказов на покупку	Быстрота	Достаточно быстрое время ответа, чтобы не мешать проведению обычных операций
Система обнаружения неполадок	Автоматическое уведомление	Все зарегистрированные стороны будут уведомлены посредством электронной почты, когда что-нибудь изменится

3.1.3 Совместная разработка приложений (JAD – Joint Application Design)

Метод совместной разработки приложений – это зарегистрированный товарный знак, относящийся к компании IBM. Он представляет собой групповой подход к определению требований, при реализации которых особое внимание уделяется усовершенствованию группового процесса и правильному подбору людей, вовлеченных в работу над проектом.

Сеансы JAD аналогичны сеансам «мозгового штурма», однако не во всем. Сеансы «мозгового штурма» длятся около двух часов, а сеансы JAD – до трех дней. На сеансах «мозгового штурма» происходит быстрое генерирование идей, а на сеансах JAD – высокоуровневые специфические программные модели функций, данных и линий поведения.

Сеанс JAD имеет определенную структуру, на нем придерживаются определенной дисциплины, и он проходит под руководством арбитра. В его основе лежит обмен информацией с использованием документации, фиксированных требований и правил работы. С момента появления методики JAD на сеансах используются CASE-инструменты и другие программные средства, предназначенные для построения диаграмм потока данных (Data flow dia- gram, DFD), диаграмм взаимосвязей между сущностями (Entity relationship diagrams, ERD), диаграмм смены состояний и других объектно-ориентированных диаграмм.

3.1.3.1 Роли в сеансах JAD

Разработчики. В задачу разработчика входит оказание помощи организаторам в формулировании их потребностей, которые обычно являются решением существующих проблем. Таким образом, определение требований к ПО происходит совместно с организаторами проекта.

Участники. Для успешного проведения сеанса ключевым требованием является высокий уровень квалификации приглашенных. Правильный подбор людей позволит быстро принимать решения и разрабатывать правильные модели, даже если они не будут завершены.

Арбитр/консультант. Арбитр должен сводить к минимуму проявление непродуктивных человеческих эмоций, наподобие агрессивности или самозащиты. Арбитр не является хозяином ни процесса, ни программного продукта. Он присутствует только для того, чтобы помогать организаторам проекта, разрабатывать программный продукт.

Секретарь. Секретарь сеанса JAD документирует идеи и помогает следить за временем.

3.1.3.2 Сеансы JAD

Согласно Вуду (Wood), сеанс JAD – это своеобразная мастерская, работающая в максимально напряженном режиме, где решения принимаются совместно всеми участниками. При этом участники являются крупными специалистами в рассматриваемом вопросе.

Процесс исследования проекта разбивается на следующие этапы: поиск фактов и сбор информации, подготовка сеанса JAD, проведение самого сеанса JAD и проверка собранной

информации.

3.1.3.3 Результаты проведения сеанса JAD

Результатами проведения сеанса могут быть:
- диаграмма контекста данных;
- диаграмма потока данных первого уровня;
- глобальная модель данных – диаграмма взаимосвязей между сущностями;
- перечень первичных объектов;
- объектная модель высокого уровня;
- обязанности кандидатов и сотрудников для каждого объекта;
- перечень первичных процессов/сценарии выбора;
- другие диаграммы потока данных, диаграммы состояния, деревья альтернатив, таблицы решений;
- требования, предъявляемые к данным для каждого процесса;
- перечень допущений;
- документация по анализу или назначению открытых вопросов.

Результаты сеанса JAD используются в процессе определения требований для организации следующего этапа – создания спецификации SRS.

3.1.3.4 Недостатки метода JAD

Например, участники передают свои идеи арбитру и/или секретарю. Во избежание неверной интерпретации собранных данных, необходимо принять некоторые меры предосторожности. Использование во время сеанса автоматизированных инструментальных средств и проверка результатов всеми участниками уменьшит риск.

На сеансах JAD рассматриваются преимущественно информационные системы, в которых особое внимание уделяется элементам данных и проекту интерфейса. Есть мало информации об использовании метода JAD для определения требований, предъявляемых к системам реального времени.

На проведение трехдневного сеанса JAD с представителями всех групп организаторов проекта, каждая из которых состоит из квалифицированных специалистов, уходит немало средств. Три дня – это средняя продолжительность. Для анализа сложных вложенных систем реального времени и систем, от которых зависит человеческая жизнь, часто требуется больше времени. Если сеанс длится «до тех пор, пока не устанем», то усталость может наступить уже в тот момент, когда будут определены только сценарии выбора.

3.1.4 Раскадровка

Целью раскадровки является получение ранней реакции пользователей на предложенные концепции приложения. С ее помощью можно на самых ранних этапах жизненного цикла наблюдать реакцию пользователей, до того как концепции будут превращены в код, а во многих случаях даже до разработки подробной спецификации.

Раскадровка имеет следующие преимущества:
- предельно недорога;
- дружественна пользователю, неформальна и интерактивна;
- обеспечивает ранний анализ пользовательских интерфейсов системы;
- легко создаваема и модифицируема.

Раскадровки можно использовать для ускорения концептуальной разработки различных граней приложения. Их можно применять для понимания визуализации данных, определения и понимания бизнес-правил, которые будут реализованы в новом бизнес-приложении, для определения алгоритмов и других математических конструкций, которые будут выполняться внутри встроенных систем, или для демонстрации отчетов и других результатов на ранних этапах. Раскадровки можно (и нужно!) использовать практически для всех приложений, в которых раннее получение реакции пользователей является ключевым фактором успеха.

3.1.4.1 Типы раскадровок

Раскадровки делятся на три типа в зависимости от режима взаимодействия с пользователем: пассивные, активные и интерактивные.

Пассивные представляют собой историю, рассказываемую пользователю. Они могут состоять из схем, картинок, моментальных копий экрана, презентаций PowerPoint или образцов выходной информации системы. В пассивной раскадровке аналитик играет роль системы и просто проводит пользователя по раскадровке, объясняя следующее: «Когда вы делаете это, происходит вот это».

Активные раскадровки обеспечивают автоматизированное описание *п*оведения системы при типовом использовании или в операционном сценарии. Они создаются с помощью анимации или автоматизации, возможно, посредством автоматического последовательного показа слайдов, анимационных средств или даже фильма.

Интерактивные дают пользователю опыт обращения с системой почти такой же реальный, как на практике. Для своего выполнения они требуют участия пользователя. Интерактивные раскадровки могут быть имитационными, в виде макетов или могут даже представлять собой отбрасываемый впоследствии код. Сложная интерактивная раскадровка, основанная на отбрасываемом коде, может быть весьма похожа на отбрасываемый прототип.

Как видно из рисунка 3.2, эти три типа раскадровки предлагают широкий спектр возможностей – от образцов выходной информации до «живых» демонстрационных версий. Различие между сложными раскадровками и ранними прототипами продукта весьма условно.

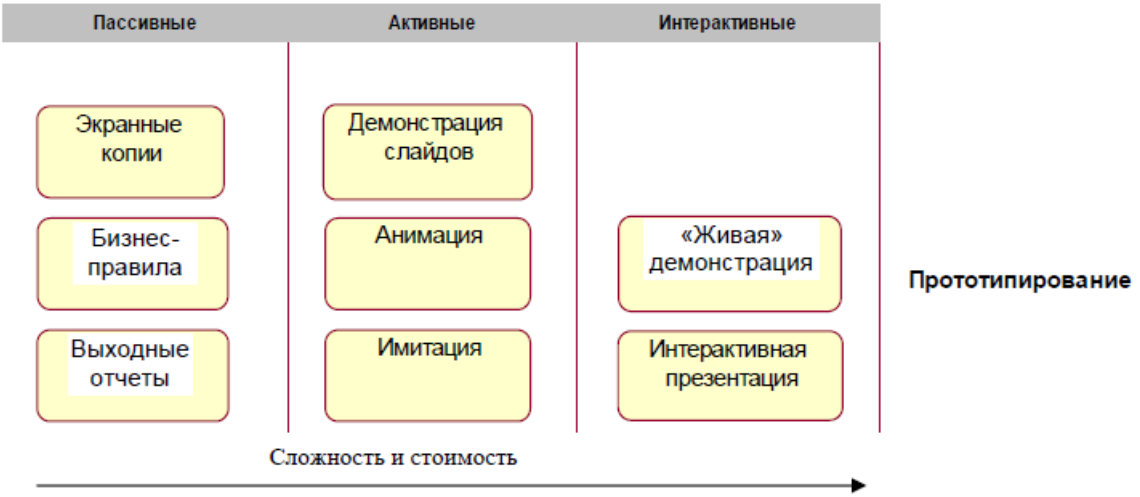


Рисунок 3.2 - Различные виды раскадровок

Выбор типа раскадровки зависит от сложности системы и того, насколько велик риск, что команда неправильно понимает ее назначение. Для беспрецедентной новой системы, имеющей расплывчатое определение, может потребоваться несколько раскадровок, от пассивной до интерактивной, по мере того как команда совершенствует свое понимание системы.

3.1.5 Обыгрывание ролей

Метод обыгрывания ролей позволяет команде разработчиков прочувствовать мир пользователя, побывав в его роли. Концепция, лежащая в основе данного метода, достаточно проста: хотя, наблюдая и задавая вопросы, мы повышаем уровень своего понимания, наивно полагать, что посредством одного наблюдения разработчик/аналитик может получить истинно глубокое понимание решаемой проблемы или требований к системе, которая призвана данную проблему решить.

3.1.5.1 Суть метода обыгрывания ролей

Аналитик или любой член команды занимает место пользователя и выполняет его

обычные действия. Рассмотрим в качестве примера проблему ввода заказов на покупку.

Разработчик/аналитик может «прочувствовать» проблемы и неточности, присущие существующей системе ввода заказов на покупку, просто заняв место оператора и попытавшись ввести несколько заказов. Полученный в течение часа опыт навсегда изменит понимание командой сути проблемы.

Существует две разновидности метода обыгрывания ролей: сценарный просмотр и CRC-карточки.

3.1.5.2 Сценарный просмотр

Сценарный просмотр – это исполнение роли на бумаге.

При сценарном просмотре каждый участник следует сценарию, который задает конкретную роль в «пьесе». Просмотр будет демонстрировать любые неточности в понимании ролей, недостаток информации, доступной актеру или подсистеме, или недостаток конкретного описания поведения, необходимого актерам для успешного выполнения их роли.

Одним из преимуществ сценарного просмотра является то, что сценарий можно модифицировать и проигрывать снова столько раз, сколько необходимо, пока актеры не сочтут его правильным. Сценарий можно также повторно использовать для обучения новых членов команды. Его можно модифицировать и проигрывать вновь, когда необходимо изменить поведение системы. В определенном смысле данный сценарий становится «живой» раскадровкой для проекта.

3.1.6 CRC-карточки (Class – Responsibility – Collaboration, класс – обязанность – взаимодействие)

Этот метод обыгрывания ролей часто применяется в объектно- ориентированном анализе. В данном случае каждому участнику выдается набор индексных карточек, описывающих класс (объект); обязанности (поведение); а также взаимодействия (с какими из моделируемых сущностей взаимодействует объект). Эти взаимодействия могут просто представлять сущности проблемной области (например, пользователи, нажатые кнопки, лампы и подъемники) или объекты, существующие в области решения (подсветка выключателя в холле, окно многодокументного интерфейса или кабина лифта).

Когда актер-инициатор инициирует определенное поведение, все участники следуют поведению, заданному на их карточках. Если процесс прерывается из-за недостатка информации или если одной сущности необходимо переговорить с другой, а взаимодействие не определено, то карточки модифицируются, и роли разыгрываются снова.

Ниже предлагается образец проигрывания одного из возможных вариантов использования.

1. **Управление включением.** Мой домовладелец только что нажал кнопку, управляющую набором ламп. Он все еще удерживает кнопку в нажатом состоянии. Я послал Центральному блоку управления сообщение, как только кнопка была нажата, и собираюсь посылать ему сообщения каждую секунду, пока кнопка нажата.

2. **Центральный блок управления.** Когда я получил первое сообщение, то изменил состояние выхода с «Выкл.» на «Вкл.». Когда я получил второе сообщение, стало очевидно, что домовладелец хочет изменить яркость освещения, поэтому при получении каждого сообщения я собираюсь изменять яркость на 10 %.

3. **Лампа.** Я аппаратно запрограммирован на изменяемый накал. Я изменяю яркость света при нашем разговоре.

3.1.7 Быстрое прототипирование

Быстрое прототипирование – это частичная реализация системы программного обеспечения, созданная с целью помочь разработчикам, пользователям и клиентам лучше понять требования к системе.

Чем детальнее прототип, тем легче понять требования заказчика. С другой стороны, прототипы сами по себе являются программами, поэтому, чем детальнее прототип, тем он дороже. Первый взгляд на проблему решения, строить прототип или нет, представлен на рисунке 3.3. Таблица показывает, например, что относительно недорогой прототип с большой

ценностью должен быть создан. Под большой ценностью имеется в виду, что построение прототипа поможет заказчику лучше понять, продукт какого типа будет выпущен, помогает программистам лучше понять, какой продукт они должны выпустить.

	Предполагаемая ценность прототипа	
	Низкая	Высокая
Низкая стоимость прототипа	Возможно	Да
Высокая стоимость прототипа	Нет	Возможно

Рисунок 3.3 - Выгода от прототипа: грубая оценка

Для случаев, когда однозначно нельзя определить, разрабатывать прототип или нет, нужно оценить затраты на разработку прототипа и возможную прибыль от его реализации. На основе этой оценки определяются оптимальные расходы на прототип и принимается решение о его разработке и размере финансирования в случае положительного решения (рисунок 3.4). По мере того как возрастают расходы на прототип, возрастает и его пригодность, но также возрастают и расходы из выделенного бюджета. В результате, вероятно, существует момент, в который затраты оптимальны (точка максимума на кри вой), и некоторая точка, за которой деньги уже потрачены зря (где кривая пересекает горизонтальную ось).

Существует много способов разбиения прототипов на категории. Выделяются следующие прототипы: отбрасываемые, эволюционирующие и операционные; вертикальные и горизонтальные; пользовательские интерфейсы и алгоритмические и т. д. Каким должен быть прототип в каждом конкретном случае, зависит от того, какую проблему вы пытаетесь решить путем построения прототипа.

Как пример представьте себе приложение для электронной коммерции, в котором компания-производитель одежды желает продавать товары через Интернет, хранить информацию о клиентах и предоставлять клиентам возможность получать свое фото в одежде из каталога. Финансовая оценка для разных уровней прототипирования для программы, продающей одежду, приведена в таблице 3.3. Для каждой из четырех характеристик, рассмотренных в прототипе, сделано несколько оценок: стоимость работы; процент работы, который будет повторно использоваться в самой программе (т. е. не будет отброшен); и полная прибыль от прототипа. Под полной прибылью здесь понимается оценка того, что будет получено, если характеристика будет включена в прототип, но код не будет использован в программе. Например, мы подсчитали, что если прототип «примерка одежды» будет построен, это сэкономит минимум 20 000 при разработке. Оценка базируется на нижеследующих факторах.

Предотвращение пустой траты времени на предложенные требования, которые, как видно из прототипа, не нужны (т. е. минимум три ненужных требования из 100; на этап требований выделено 300 000, сэкономлено 9000).

Разработка программного проекта «примерка одежды», что уменьшает риски разработки (т. е. оценка того, что это сэкономит минимум одну человеко-неделю времени проектирования = 2000).

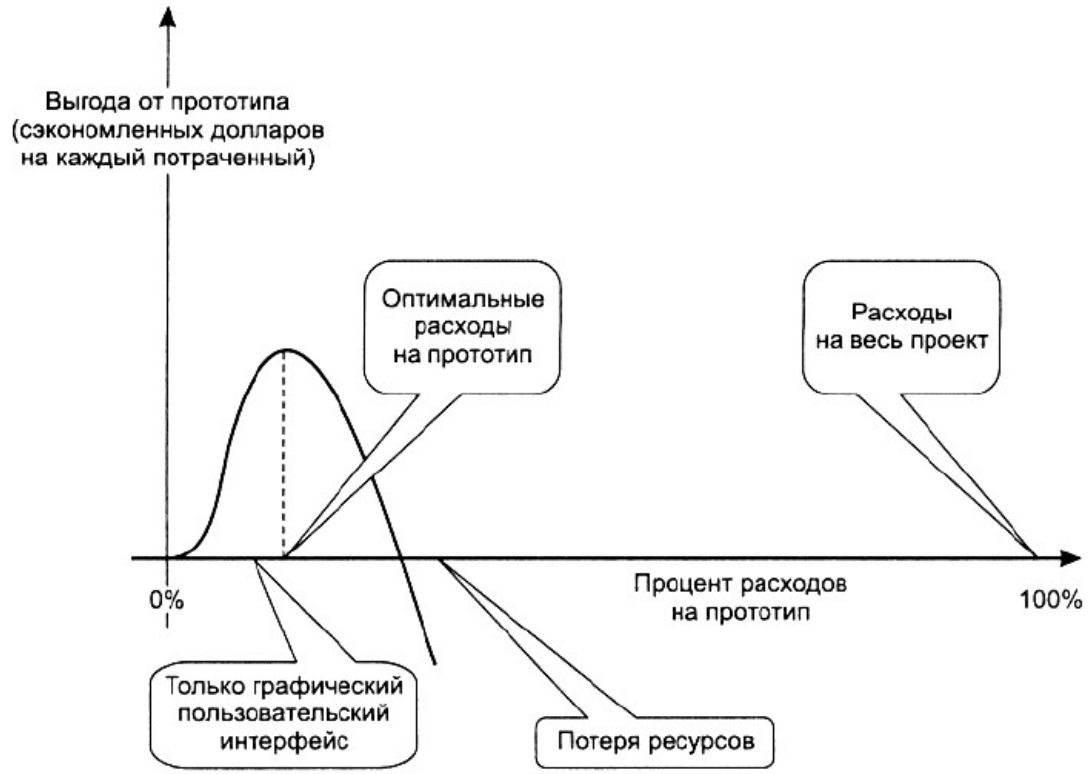


Рисунок 3.4 - Выгода от прототипа

Переработка, которая может возникнуть из-за изменения требований клиентом после того, как он увидит окончательный продукт (т. е. переработка минимум трех требований по 3000 каждое = 9000).

Существует минимальная экономия, эквивалентная 9000 + 2000 + 9000 = 20 000.

Оценка повторного использования кода может быть выполнена путем определения классов прототипа и решения того, какие из них будут использованы в самой программе.

Такая оценка состоит из исследования стоимости небольших частей, что все же является сложной задачей. Определение минимума и максимума такой оценки может несколько упростить этот трудный процесс.

Как только оценки сделаны, выполняется несложная часть вычисления лучшего и худшего сценария (таблица 3.3). Минимальное значение выгоды получается из самой пессимистичной комбинации – высоких затрат, низкой общей прибыли и низкого процента повторного использования. Максимальная выгода рассчитывается аналогично.

Усреднение – это один из способов работы с разницей между худшими и лучшими случаями. В результате получаем положительную выгоду для всех предложенных частей прототипа за исключением части «примерка одежды», которая оценена в –4000: общий убыток 4000. Это приведет в дальнейшем к относительно низкой прибыли, высокой стоимости разработки и низкому вторичному использованию.

Таблица 3.3 - Оценка программы по продаже одежды

Часть прототипа	Оценка стоимости	Общая прибыль без повторного использования кода		Процентная часть кода прототипа, повторно использованная в приложении	Чистая выгода		
		min	max		min	max	В среднем
	B	D	E	C	D – (1 – – C)B	E – (1 – – C)B	
1. Экранные снимки пользовательского интерфейса	10 000	10 000	80 000	50 %	5 000	75 000	40 000
2. Безопасность транзакций	50 000	10 000	300 000	80 %	0	290 000	145 000
3. Завершение транзакций	80 000	10 000	400 000	50 %	–30 000	200 000	85 000
4. Примерка одежды клиентом	120 000	20 000	140 000	30 %	–64 000	56 000	–4 000

3.2 Формализация требований

Иногда присущая естественному языку неоднозначность просто неприемлема, особенно когда требования касаются жизненно важных вопросов или когда неправильное поведение системы может привести к чрезвычайным экономическим или юридическим последствиям. Если определение требования сложно сформулировать на естественном языке и невозможно предотвратить неправильное понимание спецификации, следует попытаться написать эту часть требований с помощью теоретически обоснованных формальных методов.

3.2.1 Метод вариантов использования и его применение

Метод вариантов использования (прецедентов) является частью методологии объектно-ориентированного проектирования. Это метод анализа и проектирования сложных систем, представляющий собой способ описания поведения системы с точки зрения того, как различные пользователи взаимодействуют с ней для достижения своих целей. Такой ориентированный на пользователя подход предоставляет возможность исследовать различные варианты поведения системы при раннем привлечении пользователя.

Варианты использования можно успешно применять на протяжении всего жизненного цикла программного обеспечения: при анализе, проектировании и в процессе тестирования. В этом случае, процесс разработки программного обеспечения называют «основанный на вариантах использования».

Вариант использования – это функциональный связный блок, выраженный в виде транзакции между актантом и системой. Вариант использования описывает последовательность действий, выполняемых системой с целью предоставить полезный результат конкретному актанту.

3.2.1.1 Построение модели вариантов использования

Модель вариантов использования системы состоит из всех актантов системы и различных вариантов использования, посредством которых актанты взаимодействуют с системой, тем самым, описывая многообразие ее функционального поведения. Она также показывает связи между вариантами использования, что углубляет наше понимание системы.

Первый шаг моделирования вариантов использования состоит в создании системной диаграммы, описывающей границы системы и определяющей ее актанты. Это позволяет параллельно осуществить этапы 3 и 4, в которых требуется выявить заинтересованных лиц системы и определить ее границы.

Второй шаг описывает системное поведение, т. е. то, как пользователи взаимодействуют с системой, осуществляя некоторые последовательности действий, для достижения определенных

целей (рисунок 3.5).

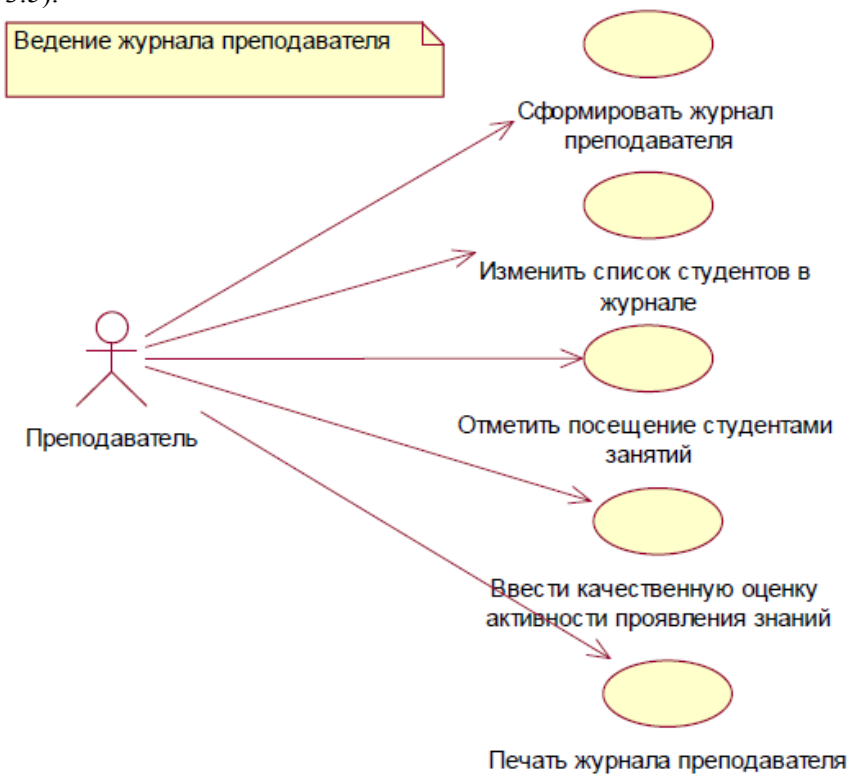


Рисунок 3.5 - Диаграмма вариантов использования

Следующий шаг состоит в уточнении деталей функционального поведения каждого варианта использования. Спецификации вариантов использования состоят из текстовых и графических описаний каждого варианта использования, написанных с позиции пользователя.

3.2.1.2 Спецификация вариантов использования

Определение потока событий. Сердцевиной варианта использования является поток событий. Это текстовое описание операций актанта и различных ответов системы. Поток событий описывает, что, как предполагается, будет делать система в зависимости от поведения актанта. Не обязательно описывать поток в текстовой форме. Для этого можно использовать диаграммы взаимодействия UML, а также другие формальные методы. Главное – обеспечить понимание, так как единственного подхода на все случаи жизни не существует. Однако, как правило, вполне подходит описание на естественном языке.

Поток событий описывает достижение цели варианта использования и предназначен для рассмотрения следующими заинтересованными лицами:

- клиентами, которые одобряют результат;
- пользователями, которые будут работать с системой;
- разработчиками вариантов использования, которые заинтересованы в точном описании желаемого поведения системы;
- рецензентами, которые составляют непредвзятое мнение о системе;
- проектировщиками, которые анализируют варианты использования в поисках классов, объектов и т. п.;
- тестерами, которым нужно создать тестовые примеры;
- менеджером проекта, которому необходимо понимать весь проект в целом;
- составителем технической документации, которому нужно документировать функции системы так, чтобы было понятно пользователю;
- людьми из отдела маркетинга и продаж, которым необходимо понимать функции продукта и объяснять его достоинства остальным.

Альтернативный поток событий. Вариант использования может иметь различные потоки в зависимости от возникающих условий. Иногда эти потоки связаны с выявленными в процессе обработки ошибочными условиями, в других случаях они могут описывать дополнительные

способы обработки конкретных условий.

В нашем примере альтернативный поток событий возникает, когда Жилец удерживает кнопку пульта в нажатом состоянии дольше одной секунды. Нам нужно добавить в вариант использования альтернативный поток.

Выявление пред- и постусловий. Использовать пред- и постусловия нужно только тогда, когда необходимо прояснить поведение, выраженное вариантом использования.

Важно проводить различие между событиями, которые запускают потоки варианта использования, и предусловиями, которые должны быть выполнены до того, как можно будет инициировать вариант использования. Например, предусловием для варианта использования «Управление освещением» является то, что домовладелец (Жилец) должен обеспечить наличие определенного набора осветительных приборов, способных изменять яркость.

Еще одним предусловием является то, что выбранная кнопка пульта должна быть запрограммирована для управления этим набором. (Предполагается, что другие варианты использования описывают, как осуществляются эти предусловия.) Итак, нам нужно сформулировать предусловия.

Постусловия позволяют точно указывать состояние, которое должно быть истинным по окончании варианта использования, даже если использовались альтернативные пути.

Чтобы яркость была такой, как нужно, когда Жилец включает свет в следующий раз, система должна «помнить» уровень яркости, который был установлен для выбранной кнопки пульта после действий по изменению яркости. Следовательно, это постусловие, которое мы должны записать для данного варианта использования.

3.2.1.3 Преимущества

Применение вариантов использования имеет ряд преимуществ по сравнению с традиционным подходом, когда определяются отдельные декларативные программные требования:

- варианты использования относительно легко писать;
- варианты использования пишутся на языке пользователя;
- варианты использования предлагают связные сценарии поведения, которые понятны как пользователю, так и разработчику.

Благодаря описанию «сценариев поведения», содержащимся в языке UML специализированным элементам и нотациям моделирования, варианты использования обеспечивают дополнительные возможности связывать деятельность по разработке требований с проектированием и реализацией.

Графическое представление вариантов использования в UML и их поддержка различными инструментальными средствами моделирования обеспечивают визуализацию связей между вариантами использования, что может содействовать пониманию сложной программной системы.

Сценарий, описанный с помощью варианта использования, может практически без изменений применяется в качестве сценария тестирования во время проверки правильности.

3.2.2 Псевдокод

Псевдокод – это квазиязык программирования; попытка соединить неформальный естественный язык со строгими синтаксическими и управляющими структурами языка программирования. В чистом виде псевдокод состоит из комбинации следующих элементов.

- Императивных предложений с одним глаголом и одним объектом.
- Ограниченного множества (как правило, не более 40–50) «ориентированных на действия» глаголов, из которых должны конструироваться предложения.
- Решений, представленных формальной структурой IF-ELSE-ENDIF.
- Итеративных действий, представленных структурами DO-WHILE и FOR-NEXT.

На рисунке 3.6 представлен пример спецификации с помощью псевдокода алгоритма вычисления отложенного дохода от услуг в течение данного месяца в бизнес-приложении.

```
Set Sum(X)=0
for каждого клиента x
   if клиент оплатил услуги вперед
        and ((Текущий месяц)>=(2 мес. после даты приобретения)) and ((Текущий
        месяц)<=(14 мес. после даты приобретения))
   then Sum(X) = Sum(x) + (сумма, заплаченная клиентом)/12
```

Рисунок 3.6 - Пример спецификации с использованием псевдокода

Фрагменты текста на псевдокоде расположены уступами; такой формат используется для того, чтобы выделить логические блоки. Сочетание синтаксических ограничений, формата и разбивки существенно уменьшает неоднозначность требования, которое в противном случае было бы очень сложным и расплывчатым. В то же время такое представление требования вполне понятно для человека, не являющегося программистом. Не нужно быть выдающимся ученым, чтобы понимать псевдокод, и нет необходимости знать C++ или Java.

3.2.3 Конечные автоматы

Представление в виде конечного автомата описывает возможные жизненные циклы объекта и состоит из состояний, соединенных переходами.

Каждое состояние – это такой период жизненного цикла объекта, когда он удовлетворяет определенным условиям. Некоторое событие может привести к переходу, в результате которого объект окажется в новом состоянии.

При переходе может выполняться действие, предписанное данному переходу.

Представление в виде конечного автомата изображается на диаграммах состояний и переходов.

На рисунке 3.7 приведен пример диаграммы состояний и переходов для объекта «Заказ».

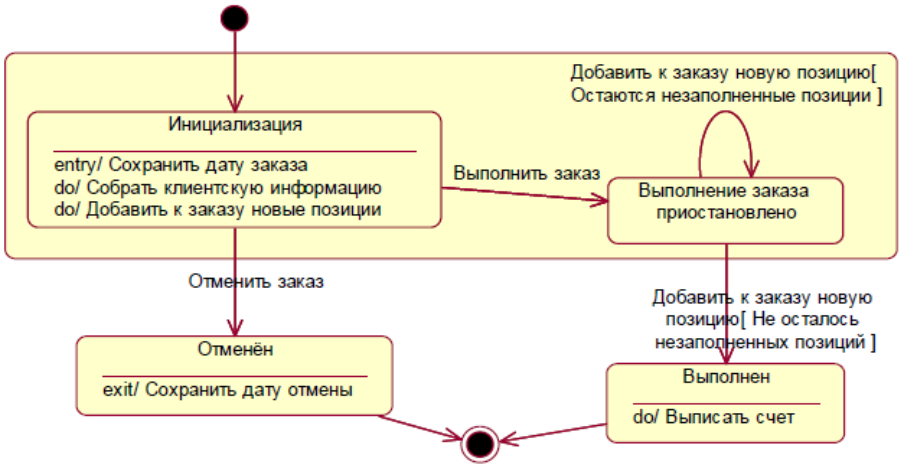


Рисунок 3.7 - Диаграмма состояний и переходов объекта «Заказ»

3.2.4 Графические деревья решений

Дерево решений применяется для отображения информации в виде графа. На рисунке 3.8 показано дерево решений, используемое для описания последовательности действий.

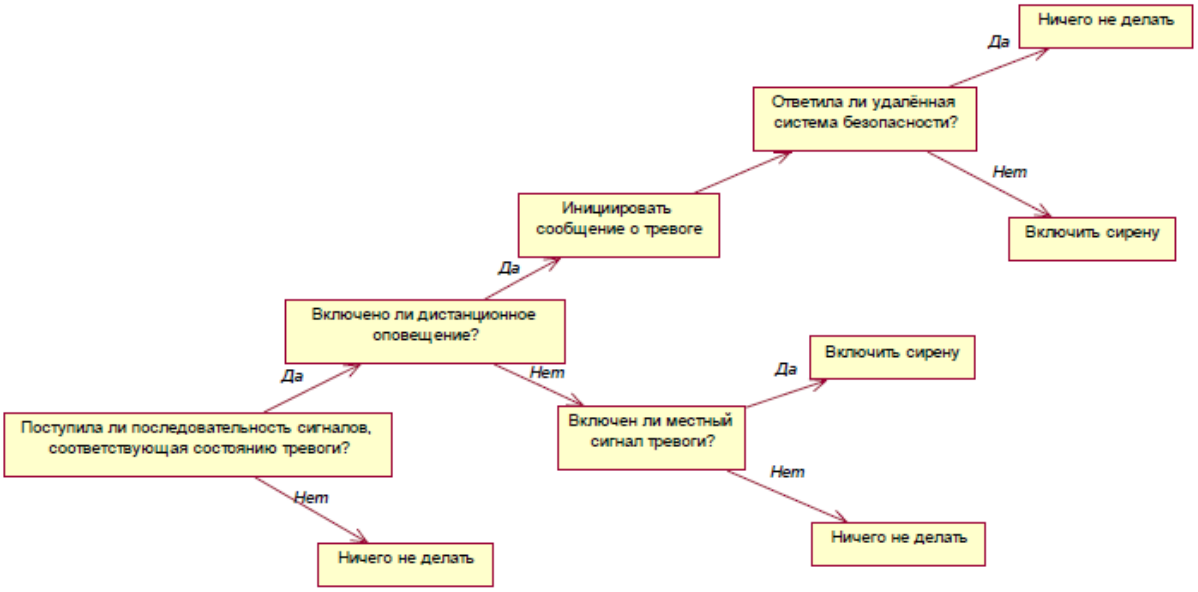


Рисунок 3.8 - Графическое дерево решений

3.2.5 Диаграммы деятельности

Деятельность представляет собой поток работ или выполнение операции. Представление деятельности отображает как последовательные, так и параллельные виды деятельности. Изображаются такие модели на диаграммах деятельности.

На рисунке 3.9 изображена диаграмма, которая описывает деятельности покупателя в Интернет-магазине. Здесь представлены две точки ветвления – для выбора способа поиска товара и для принятия решения о покупке. Присутствуют три линейки синхронизации: верхняя – отражает разделение на два параллельных процесса, средняя отражает и разделение, и слияние процессов, а нижняя – только слияние процессов. Дополнительно на этой диаграмме показаны две дорожки – дорожка покупателя и дорожка магазина, которые разделены вертикальной линией. Каждая дорожка имеет имя и фиксирует область деятельности конкретного лица, обозначая зону его ответственности.

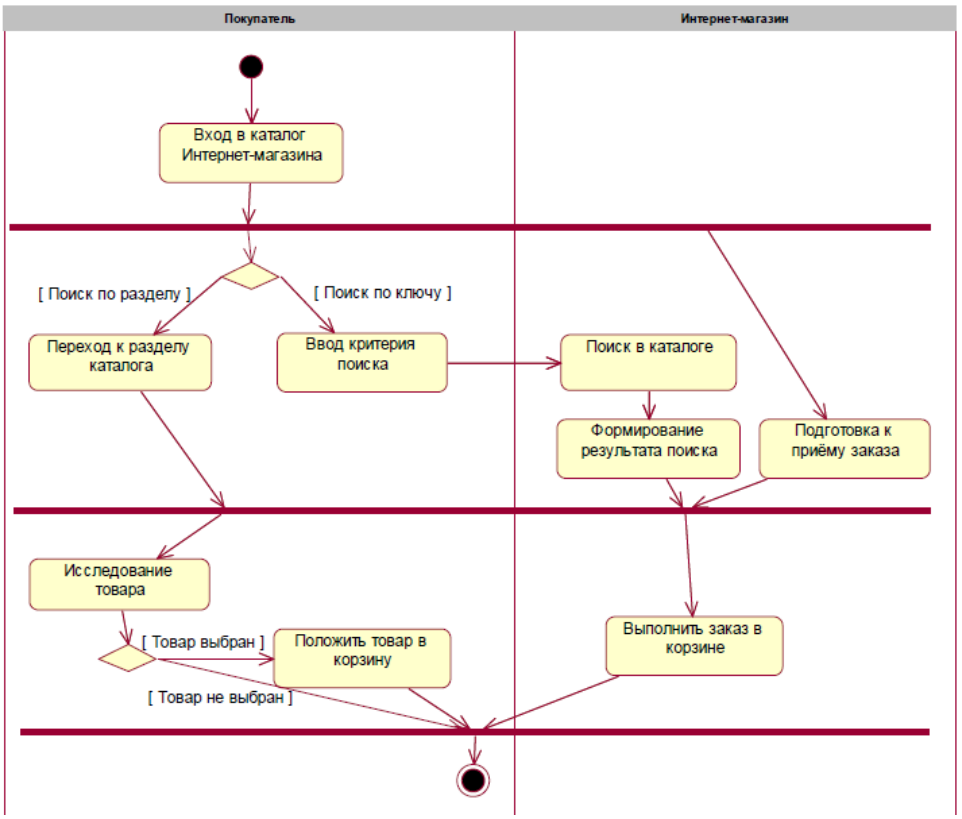


Рисунок 3.9 - Диаграмма деятельности покупателя в Интернет-магазине

Диаграмма деятельности отражает реальные потоки работ в человеческой организации. Такое бизнес-моделирование и есть основное ее назначение. С таким же успехом ее можно использовать и при моделировании работ программного приложения.

3.3 Техническое задание (ГОСТ34.602–89)

Настоящий стандарт распространяется на автоматизированные системы (АС) для автоматизации различных видов деятельности (управление, проектирование, исследование и т. п.), включая их сочетания, и устанавливает состав, содержание, правила оформления документа «Техническое задание на создание (развитие или модернизацию) системы».

3.3.1 Общие сведения

В этот раздел включают: полное наименование системы и ее условное обозначение; шифр темы или шифр (номер) договора; наименование предприятий (объединений) разработчика и заказчика (пользователя) системы и их реквизиты; перечень документов, на основании которых создается система, кем и когда утверждены эти документы; плановые сроки начала и окончания работы по созданию системы; сведения об источниках и порядке финансирования работ; порядок оформления и предъявления заказчику результатов работ по созданию системы (ее частей), по изготовлению и наладке отдельных средств (технических, программных, информационных) и программно-технических (программно-методических) комплексов системы.

3.3.2 Назначение и цели создания (развития) системы
3.3.2.1 Назначение системы

Здесь указывают вид автоматизируемой деятельности (управление, проектирование и т. п.) и перечень объектов автоматизации, на которых предполагается ее использовать. Для АСУ дополнительно указывают перечень автоматизируемых органов (пунктов) управления и управляемых объектов.

3.3.2.2 Цели создания системы

Приводят наименования и требуемые значения технических, технологических, производственно-экономических или других показателей объекта автоматизации, которые должны быть достигнуты в результате создания АС, и указывают критерии оценки достижения целей создания системы.

3.3.3 Характеристики объекта автоматизации

В данном разделе приводят: краткие сведения об объекте автоматизации или ссылки на документы, содержащие такую информацию; сведения об условиях эксплуатации объекта автоматизация и характеристиках окружающей среды.

Примечание. Для САПР дополнительно приводят основные параметры и характеристики объектов проектирования.

3.3.4 Требования к системе

Состав требований к системе, включаемых в данный раздел ТЗ на АС, устанавливают в зависимости от вида, назначения, специфических особенностей и условий функционирования конкретной системы. В каждом подразделе приводят ссылки на действующие НТД, определяющие требования к системам соответствующего вида.

3.3.4.1 Требования к системе в целом

Требования к структуре и функционированию системы. Здесь приводят: перечень

подсистем, их назначение и основные характеристики, требования к числу уровней иерархии и степени централизации системы; требования к способам и средствам связи для информационного обмена между компонентами системы; требования к характеристикам взаимосвязей создаваемой системы со смежными системами, требования к ее совместимости, в том числе указания о способах обмена информацией (автоматически, пересылкой документов, по телефону и т. п.); требования к режимам функциони- рования системы; требования по диагностированию системы; перспективы развития, модернизации системы.

Требования к численности и квалификации персонала на АС. Приводят: требования к численности персонала (пользователей) АС; требования к квалификации персонала, порядку его подготовки и контроля знаний и навыков; требуемый режим работы персонала АС.

Требования к показателям назначения АС. Приводят значения параметров, характеризующих степень соответствия системы ее назначению.

Для АСУ указывают: степень приспособляемости системы к изменению процессов, к отклонениям параметров объекта управления; номер методов управления; допустимые пределы модернизации и развития системы; вероятностно-временные характеристики, при которых сохраняется целевое назначение системы.

Требования к надежности. Включают: состав и количественные значения показателей надежности для системы в целом или ее подсистем; перечень аварийных ситуаций, по которым должны быть регламентированы требования к надежности, и значения соответствующих показателей; требования к надежности технических средств и программного обеспечения; требования к методам оценки и контроля показателей надежности на разных стадиях создания системы в соответствии с действующими нормативно-техническими документами.

Требования по безопасности. Включают требования по обеспечению безопасности при монтаже, наладке, эксплуатации, обслуживании и ремонте технических средств системы (защита от воздействий электрического тока, электромагнитных полей, акустических шумов и т. п.), по допустимым уровням освещенности, вибрационных и шумовых нагрузок.

Требования по эргономике и технической эстетике. Включают показатели АС, задающие необходимое качество взаимодействия человека с машиной и комфортность условий работы персонала.

Требования к эксплуатации, техническому обслуживанию, ремонту и хранению. Включают: условия и регламент (режим) эксплуатации, которые должны обеспечивать использование технических средств (ТС) системы с заданными техническими показателями, в том числе виды и периодичность обслуживания ТС системы или допустимость работы без обслуживания; предварительные требования к допустимым площадям для размещения персонала и ТС системы, к параметрам сетей энергоснабжения и т. п.; требования по количеству, квалификации обслуживающего персонала и режимам его работы; требования к составу, размещению и условиям хранения комплекта запасных изделий и приборов; требования к регламенту обслуживания.

Требования к защите информации от несанкционированного доступа. Включают требования, установленные в НТД, действующей в отрасли (ведомстве) заказчика.

Требования по сохранности информации. Приводят перечень событий: аварий, отказов технических средств (в том числе потеря питания) и т. п., при которых должна быть обеспечена сохранность информации в системе.

Требования к средствам защиты от внешних воздействий. Приводят: требования к радиоэлектронной защите средств АС; требования по стойкости, устойчивости и прочности к внешним воздействиям (среде применения).

Требования к патентной чистоте. Указывают перечень стран, в отношении которых должна быть обеспечена патентная чистота системы и ее частей.

Требования к стандартизации и унификации. Включают: показатели, устанавливающие требуемую степень использования стандартных, унифицированных методов реализации функций (задач) системы, поставляемых программных средств, типовых математических методов и моделей, типовых проектных решений, унифицированных форм управленческих документов, установленных ГОСТ 6.10.1, общесоюзных классификаторов технико-экономической информации и классификаторов других категорий в соответствии с областью их применения, требования к использованию типовых автоматизированных рабочих мест, компонентов и комплексов.

Дополнительные требования. Включают: требования к оснащению системы

устройствами для обучения персонала (тренажерами, другими устройствами аналогичного назначения) и документацией на них; требования к сервисной аппаратуре, стендам для проверки элементов системы; требования к системе, связанные с особыми условиями эксплуатации; специальные требования по усмотрению разработчика или заказчика системы.

3.3.4.2 Требования к функциям (задачам)

Здесь приводят: по каждой подсистеме перечень функций, задач или их комплексов (в том числе обеспечивающих взаимодействие частей системы), подлежащих автоматизации; при создании системы в две или более очереди – перечень функциональных подсистем, отдельных функций или задач, вводимых в действие в 1-й и последующих очередях; временной регламент реализации каждой функции, задачи (или комплекса задач); требования к качеству реализации каждой функции (задачи или комплекса задач), к форме представления выходной информации, характеристики необходимой точности и времени выполнения, требования одновременности выполнения группы функций, достоверности выдачи результатов; перечень и критерии отказов для каждой функции, по которой задаются требования по надежности.

3.3.4.3 Требования к видам обеспечения

В зависимости от вида системы приводят требования к математическому, информационному, лингвистическому, программному, техническому, метрологическому, организационному, методическому и другие видам обеспечения системы.

Требования к математическому обеспечению системы. Приводят требования к составу, области применения (ограничения) и способам использования в системе математических методов и моделей, типовых алгоритмов и алгоритмов, подлежащих разработке.

Требования к информационному обеспечению. Приводят требования: к составу, структуре и способам организации данных в системе; к информационному обмену между компонентами системы; к информационной совместимости со смежными системами; по использованию общесоюзных и зарегистрированных республиканских, отраслевых классификаторов, унифи- цированных документов и классификаторов, действующих на данном предприятии; по применению систем управления базами данных; к структуре процесса сбора, обработки, передачи данных в системе и представлению данных; к защите данных от разрушений при авариях и сбоях в электропитании системы; к контролю, хранению, обновлению и восстановлению данных; к процедуре придания юридической силы документам, продуцируемым техническими средствами АС (в соответствии с ГОСТ 6.10.4).

Требования к лингвистическому обеспечению. Приводят требования к применению в системе языков программирования высокого уровня, языков взаимодействия пользователей и технических средств системы, а также требования к кодированию и декодированию данных, к языкам ввода-вывода данных, языкам манипулирования данными, средствам описания предметной области (объекта автоматизации), к способам организации диалога.

Требования к программному обеспечению. Приводят перечень покупных программных средств. А также требования: к независимости программных средств от используемых комплексов технических средств и операционной среды; к качеству программных средств, а также к способам его обеспечения и контроля; по необходимости согласования вновь разрабатываемых программных средств с фондом алгоритмов и программ.

Требования к техническому обеспечению. Приводят требования: к видам технических средств, в том числе к видам комплексов технических средств, программно-технических комплексов и других комплектующих изделий, допустимых к использованию в системе; к функциональным, конструктивным и эксплуатационным характеристикам средств технического обеспечения системы.

Требования к метрологическому обеспечению. Приводят: предварительный перечень измерительных каналов; требования к точности измерений параметров и (или) к метрологическим характеристикам измерительных каналов; требования к метрологической совместимости технических средств системы; перечень управляющих и вычислительных каналов системы, для которых необходимо оценивать точностные характеристики; требования к метрологическому обеспечению технических и программных средств, входящих в состав измерительных каналов системы, средств, встроенного контроля, метрологической пригодности

измерительных каналов и средств измерений, используемых при наладке и испытаниях системы; вид метрологической аттестации (государственная или ведомственная) с указанием порядка ее выполнения и организаций, проводящих аттестацию.

Требования к организационному обеспечению. Приводят: к структуре и функциям подразделений, участвующих в функционировании системы или обеспечивающих эксплуатацию; к организации функционирования системы и порядку взаимодействия персонала АС и персонала объекта автоматизации; к защите от ошибочных действий персонала системы.

Требования к методическому обеспечению САПР. Приводят требования к составу нормативно-технической документации системы (перечень применяемых при ее функционировании стандартов, нормативов, методик и т. п.).

3.3.5 Состав и содержание работ по созданию (развитию) системы

Содержаться перечень стадий и этапов работ по созданию системы в соответствии с ГОСТ 24.601, сроки их выполнения, перечень организаций – исполнителей работ, ссылки на документы, подтверждающие согласие этих организаций на участие в создании системы, или запись, определяющую ответственного (заказчик или разработчик) за проведение этих работ.

Кроме того приводят:

- перечень документов по ГОСТ 34.201, предъявляемых по окончании соответствующих стадий и этапов работ;
- вид и порядок проведения экспертизы технической документации (стадия, этап, объем проверяемой документации, организация-эксперт);
- программу работ, направленных на обеспечение требуемого уровня надежности разрабатываемой системы (при необходимости);
- перечень работ по метрологическому обеспечению на всех стадиях создания системы с указанием их сроков выполнения и организаций-исполнителей (при необходимости).

3.3.6 Порядок контроля и приемки системы

Указывают: виды, состав, объем и методы испытаний системы и ее составных частей (виды испытаний в соответствии с действующими нормами, распространяющимися на разрабатываемую систему); общие требования к приемке работ по стадиям (перечень участвующих предприятий и организаций, место и сроки проведения), порядок согласования и утверждения приемочной документации; статус приемочной комиссии (государственная, межведомственная, ведомственная).

3.3.7 Требования к составу и содержанию работ по подготовке объекта автоматизации к вводу системы в действие

Здесь необходимо привести перечень основных мероприятий и их исполнителей, которые следует выполнить при подготовке объекта автоматизации к вводу АС в действие.

В перечень основных мероприятий включают: приведение поступающей в систему информации (в соответствии с требованиями к информационному и лингвистическому обеспечению) к виду, пригодному для обработки с помощью ЭВМ; изменения, которые необходимо осуществить в объекте автоматизации; создание условий функционирования объекта автоматизации, при которых гарантируется соответствие создаваемой системы требованиям, содержащимся в ТЗ; создание необходимых для функционирования системы подразделений и служб; сроки и порядок комплектования штатов и обучения персонала.

Например, для АСУ приводят: изменения применяемых методов управления; создание условий для работы компонентов АСУ, при которых гарантируется соответствие системы требованиям, содержащимся в ТЗ.

3.3.8 Требования к документированию

Приводят: согласованный разработчиком и заказчиком системы перечень подлежащих разработке комплектов и видов документов, соответствующих требованиям ГОСТ 34.201 и НТД отрасли заказчика; перечень документов, выпускаемых на машинных носителях; требования

к микрофильмированию документации; требования по документированию комплектующих элементов межотраслевого применения в соответствии с требованиями ЕСКД и ЕСПД; при отсутствии государственных стандартов, определяющих требования к документированию элементов системы, дополнительно включают требования к составу и содержанию таких документов.

3.3.9 Источники разработки

Должны быть перечислены документы и информационные материалы (технико-экономическое обоснование, отчеты о законченных научно-исследовательских работах, информационные материалы на отечественные, зарубежные системы-аналоги и др.), на основании которых разрабатывалось ТЗ и которые должны быть использованы при создании системы.

План конспекта:

1. Введение в системный анализ.
2. Анализ проблемы и моделирование предметной области с использованием системного подхода.
3. Методология ARIS.
4. Стандарты IDEF0 – IDEF3.
5. Методы определения требований.
6. Формализация требований.
7. Техническое задание (ГОСТ 34.602–89).

Работа с литературой:

Рекомендуемые источники информации (№ источника)			
Основная	Дополнительная	Методическая	Интернет-ресурсы
1-3	1-3	1-3	1-2

Тема самостоятельного изучения № 4
Архитектуры программных систем.

Вид деятельности студентов: самостоятельное изучение литературы
Итоговый продукт самостоятельной работы: конспект
Средства и технологии оценки: собеседование
Краткие теоретические сведения

4 АРХИТЕКТУРЫ ПРОГРАММНЫХ СИСТЕМ

4.1 Планирование архитектуры

Современные методы разработки программного обеспечения предполагают обратную связь между всеми действующими лицами, от проектировщика до аналитика. Все эти лица являются участниками процесса создания архитектуры программной системы. Под архитектурой системы будем понимать структуру компонентов программной системы, взаимосвязи, а также принципы и нормы их проектирования и развития во времени. Прежде чем начать изучение процесса планирования архитектуры, необходимо познакомиться с понятием архитектурно-экономического цикла (АЭЦ).

4.1.1 Архитектурно-экономический цикл

Взаимоотношения между производственными задачами, требования к продукту, опыт архитектора, архитектуры и созданные системы образуют цикл с цепями обратной связи. Упомянутые цепи обратной связи изображены на рисунке 4.1. Частично обратная связь поступает от самой архитектуры, частично – от построенной на ее основе системы.

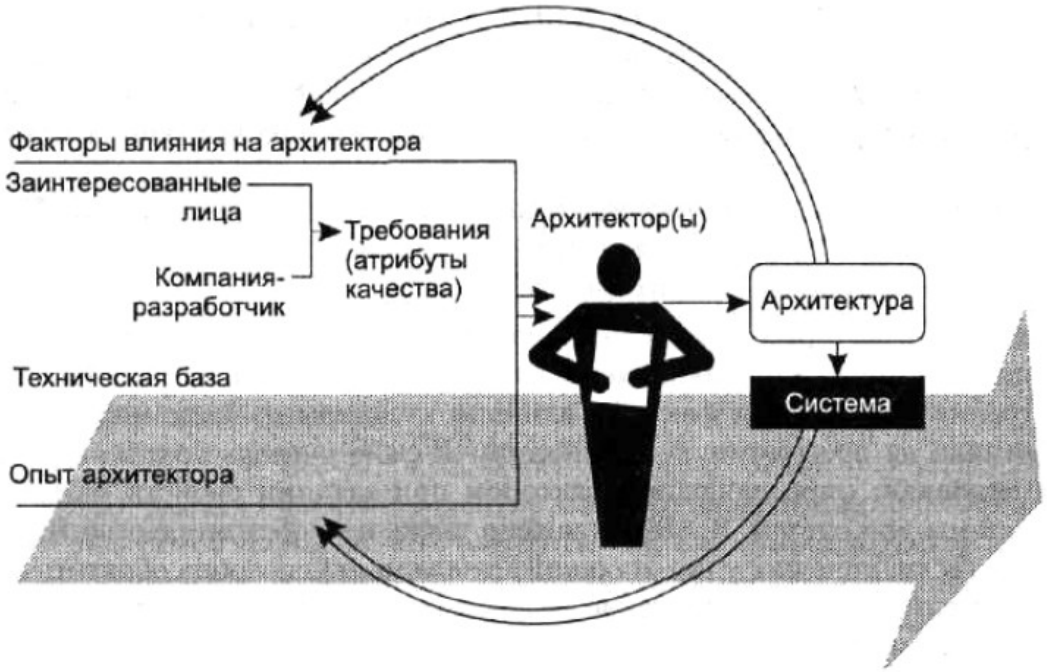


Рисунок 4.1 Архитектурно-экономический цикл

Цикл выглядит следующим образом.

1. Архитектура влияет на структуру компании-разработчика. Архитектура обусловливает структуру системы; в частности (в этом мы сможем убедиться), она устанавливает набор блоков программного обеспечения, которое надлежит реализовать (или обеспечить их наличие другим путем), а затем интегрировать в рамках системы. Эти блоки составляют основу разработки структуры проекта. Группы разработчиков укомплектовываются именно по блокам; операции в рамках процессов разработки, тестирования и интеграции также выполняются в отношении блоков. Согласно графикам и бюджетам, ресурсы выделяются частями в расчете на отдельные блоки. Если компания наработала опыт конструирования семейств сходных систем, она будет вкладывать средства в повышение профессионального уровня участников сформированных по блокам групп разработчиков. Следовательно, группы встраиваются в структуру организации. Такой представляется обратная связь от архитектуры к компании-разработчику.

2. Архитектура способна оказывать воздействие на задачи компании- разработчика. Сконструированная на ее основе успешная система предоставляет компании возможность укрепиться в данном сегменте рынка. Такая архитектура предусматривает дальнейшее эффективное производство и размещение сходных систем, вследствие чего компания может откорректировать свои задачи и, воспользовавшись новым преимуществом, занять рыночную

нишу. Так выглядит обратная связь от системы к компании-разработчику и конструируемым ею системам.

3. Архитектура может оказывать воздействие на требования, выдвигаемые заказчиком относительно следующей системы, – ее (если она основана на той же архитектуре, что и предыдущая) он может получить в более надеж ном варианте, быстрее и экономичнее, чем в том случае, если бы она конструировалась с чистого листа. Возможно, заказчик откажется от некоторых требований в пользу повышения экономичности. Готовые программные продукты несколько изменили требования, предъявляемые заказчиками, – не предназначенные для удовлетворения индивидуальных потребностей, они недороги и отличаются высоким качеством. На заказчиков, не слишком гибких по части своих требований, аналогичное воздействие оказывают линейки продуктов.

4. Процесс конструирования систем пополняет опыт архитектора, который он может применить при работе над последующими архитектурами, и, соответственно, расширяет базу опыта компании. Успех системы, построенной на основе инструментальных магистралей, .NET или инкапсулированных конечных автоматов, стимулирует построение аналогичных систем в дальнейшем. С другой стороны, неудачные варианты архитектуры редко используются повторно.

5. Иногда оказать сильное воздействие и даже внести изменения в культуру программной инженерии (техническую базу, в рамках которой обучаются и работают конструкторы) способны отдельные системы. Такой эффект на индустрию в 1960-х и начале 1970-х гг. оказали первые реляционные базы данных, генераторы компиляторов и табличные операционные системы; в 1980-х – первые электронные таблицы и системы управления окнами. В 1990-х гг. в качестве такого рода катализатора выступила Всемирная паутина. Подобные инновации всегда находят отражение в последующих системах.

Из этих и некоторых других механизмах обратной связи и образуется архитектурно-экономический цикл. На рисунке 4.1 изображены факторы влияния культуры и экономики компании-разработчика на программную архитектуру. В свою очередь архитектура оказывается основным определяющим фактором при задании свойств разрабатываемой системы или систем.

4.1.2 Программный процесс и архитектурно-экономический цикл

Программным процессом (software process) называются действия по организации, нормированию и управлению разработкой программного обеспечения. Ниже приведен перечень операций, направленных на создание программной архитектуры, ее применение для реализации проектного решения, а впоследствии – на реализацию или управление развитием целевой системы или приложения:

- создание экономической модели системы;
- выявление требований;
- создание новой или выбор существующей архитектуры;
- документирование и распространение сведений об архитектуре;
- анализ или оценка архитектуры;
- реализация системы на основе архитектуры;
- проверка соответствия реализации архитектуре.

4.1.2.1 Этапы разработки архитектуры

Как следует из структуры АЭЦ, между различными этапами разработки архитектуры существуют развернутые отношения обратной связи

Создание экономической модели системы. Создание экономической модели не ограничивается оценкой потребности в системе на рынке. Этот этап исполняет существенную роль в контексте создания и сужения требований. Сколько должен стоить продукт? Каков целевой сегмент рынка? Насколько быстро продукт должен выйти на рынок? Должен ли он взаимодействовать с другими системами? Есть ли какие-нибудь системные ограничения, в рамках которых он должен существовать? Все эти вопросы решаются с привлечением архитектора. Одного его, конечно, недостаточно, однако если при создании экономической модели консультации с архитектором не проводились, то возможность достижения коммерческих задач становится проблематичной.

Выявление требований. Способов узнать, чего же, наконец, хотят заинтересованные лица, множество. В частности, в рамках объектно-ориентированного анализа для фиксации

требований используются сценарии, или элементы Use Case. Для системы с повышенными требованиями к безопасности применяются более строгие методики – например, модели конечных автоматов и языки формальных спецификаций.

Применительно к конструируемой системе необходимо принять центральное, основополагающее решение – насколько в ней будут отражены другие, уже сконструированные системы. Поскольку в сегодняшних условиях найти систему, не имеющую сходств с другими системами, весьма непросто, методики выявления требований предполагают знание характеристик предшествующих систем.

Еще один способ выявления требований подразумевает моделирование. Опытные системы помогают моделировать нужное поведение, проектировать пользовательские интерфейсы и проводить анализ потребления ресурсов. Таким образом, в глазах заинтересованных лиц система становится «реальной», а процесс принятия решений по проектированию системы и ее пользовательского интерфейса значительно ускоряется.

Вне зависимости от методики выявления требований, желаемые атрибуты качества конструируемой системы обусловливают ее конечный вид. Для обеспечения отдельных атрибутов качества архитекторами уже давно применяются те или иные тактики. Архитектурные решения компромиссны, однако при специфицировании требований не все эти компромиссы очевидны. Со всей ясностью они проявляются только после создания архитектуры; тогда же принимаются решения относительно сортировки требований в соответствии с приоритетами.

Создание или выбор архитектуры. Фредерик Брукс (Fred Brooks) в своей знаменитой книге «Мифический человеко-месяц» красноречиво и убедительно доказывает, что основным условием стабильного проектирования системы является соблюдение концептуальной целостности, а она может проявиться лишь в узком кругу людей, совместно работающих над проектированием ее архитектуры.

Распространение сведений об архитектуре. Для того чтобы архитектура действительно стала основой проекта, ее суть необходимо четко и недвусмысленно донести до всех заинтересованных лиц. Разработчики должны понимать, что от них требуется, тестировщики должны осознавать структуру своих задач, менеджмент должен знать график и т. д. Для того чтобы этой цели можно было добиться, документирование архитектуры должно быть информативным, ясным и понятным людям различных профессий.

Анализ или оценка архитектуры. В процессе проектирования всегда рассматривается множество вариантов проекта. Некоторые из них забраковываются сразу. Из числа остальных в конечном итоге отбирается наиболее подходящий. Одна из глобальных задач, стоящих перед любым архитектором, заключается именно в том, чтобы сделать этот выбор рационально.

Оценить архитектуру на предмет атрибутов качества, которые она обеспечивает, совершенно необходимо – без этого нельзя быть уверенным в том, что конечная система сможет удовлетворить все потребности заинтересованных лиц. Все большее распространение получают методики анализа, ориентированные на оценку сообщаемых системе архитектурой атрибутов качества. Сценарные методики обеспечивают наиболее универсальную и эффективную оценку архитектуры. Самая зрелая методическая база характерна для метода анализа компромиссных архитектурных решений (Architecture Tradeoff Analysis Method, ATAM); метод анализа стоимости и эффективности (Cost Benefit Analysis Method, CBAM), с другой стороны, предусматривает крайне ценную возможность увязки архитектурных решений с их экономическим содержанием.

4.1.3 Суть программной архитектуры

Программная архитектура программы или вычислительной системы – это структура ее структур, т. е. изложение ее программных элементов, их внешних свойств и установленных между ними отношений.

Внешними (externally visible) свойствами называются те предположения, которые сторонние элементы могут выдвигать в отношении данного элемента, – в частности, они касаются предоставляемых элементом услуг, рабочих характеристик, устранения неисправностей, совместного использования ресурсов и т. д. Проанализируем представленное определение более подробно.

Во-первых, архитектура определяет программные элементы. В составе архитектуры

приводится информация о взаимоотношениях элементов. Собственно говоря, все, что архитектура сообщает нам об элементах, ограничивается информацией об их взаимодействии. Итак, архитектура – это, в первую очередь, абстракция системы, в которой отсутствует информация об элементах, не имеющая отношения к тому, как они используют, используются, соотносятся или взаимодействуют с другими элементами. Практически во всех современных системах взаимодействие между элементами осуществляется посредством интерфейсов, которые поддерживают деление информации об элементах на публичную и приватную части. Так вот, архитектура имеет дело только с публичной частью; приватные детали элементов, те, что относятся исключительно к их внутренней реализации, в состав архитектуры не входят.

Во-вторых, из вышеприведенного определения ясно, что в состав любой системы может входить и входит целый ряд структур, а следовательно, ни одной отдельно взятой структуры, которую можно было бы уверенно назвать архитектурой, не существует. В частности, все нетривиальные проекты делятся на блоки реализации; между этими блоками распределяются некоторые обязанности, и они же в большинстве случаев выступают в качестве базы для разделения задач между группами программистов. В таких элементах наряду с программами и данными, доступными для вызова или обращения со стороны других программных средств и блоков реализации, содержатся приватные программы и данные. В крупных проектах эти элементы почти всегда разделяются на мелкие части, а обязанности по работе с ними распределяются между несколькими мелкими группами разработчиков. Довольно часто описания систем составляются при помощи таких структур. Ориентированные на разделение функций системы между различными группами исполнителей реализации, они отличаются чрезмерной статичностью.

Другие структуры в большей степени ориентируются на взаимодействие элементов в период прогона с целью исполнения функции системы. Представим, что систему предполагается сконструировать в виде ряда паралельных процессов. В этом случае часто применяется другая структура, включающая процессы периода прогона, программы из вышеописанных блоков реализации, совместно формирующие отдельные процессы, а также установленные между этими процессами отношения синхронизации.

Можем ли мы взять любую из этих структур в отдельности и назвать ее архитектурой? Нет, не можем – и это несмотря на то, что все они содержат архитектурные сведения. В состав любой архитектуры эти структуры входят наравне со многими другими. В этой связи, очевидно, что, поскольку в составе архитектуры может содержаться несколько структур, в ней должны присутствовать несколько элементов (например, блок реализации и процессы), несколько вариантов взаимодействия между элементами (например, разбиение на составляющие и синхронизация) и даже несколько контекстов (например, время разработки и время прогона). Представленное определение не устанавливает сущность архитектурных элементов и отношений. Является ли программный элемент объектом? процессом? библиотекой? базой данных? коммерческим изделием? Он может быть чем угодно, причем возможности не ограничиваются вышеприведенными вариантами.

В-третьих, согласно нашему определению, программная архитектура есть у любой вычислительной системы с программным обеспечением. Связано это с тем, что любую систему можно представить как совокупность ее элементов и установленных между ними отношений. В простейшем случае система сама по себе является элементом, не представляющим, вероятно, никакого интереса и бесполезным, однако, тем не менее, согласующимся с понятием «архитектура». То обстоятельство, что архитектура есть у каждой системы, совершенно не означает, что она является общеизвестной. Кто знает, быть может, специалистов, спроектировавших систему, уже не найти, документация тоже куда-то исчезла (или ее никогда не существовало), исходный код потерян (или не поставлялся), а остался лишь исполняемый двоичный код. Именно в этом случае различие между архитектурой системы и ее представлением становится очевидным. К сожалению, архитектура иногда существует самостоятельно, без описаний и спецификаций; в этой связи существенную важность приобретают документирование и реконструкция архитектуры.

В-четвертых, если поведение отдельно взятого элемента можно зафиксировать или выявить с точки зрения других элементов, то это поведение входит в состав архитектуры. Именно оно позволяет элементам взаимодействовать друг с другом, а взаимодействие, как известно, отражается в архитектуре в обязательном порядке. Этим, помимо прочего, объясняется, почему схемы из прямоугольников и линий, выдаваемые за архитектуры, таковыми

не являются. Это не более чем рисунки; самое лучшее, что о них можно сказать, – это то, что они намекают на существование более четкой информации относительно фактических функций обозначенных элементов. Взглянув на наименования изображенных на подобной схеме прямоугольников (например, на них написано «база данных», «пользовательский интерфейс», «ис- полняемый файл» и т. д.), читатель хотя бы получит представление о функциональности и поведении соответствующих элементов.

Наконец, в представленном определении не отражено качество архитектуры системы, иначе говоря, никаких утверждений относительно перспектив соответствия системы установленным требованиям к поведению, производительности и жизненному циклу в ней нет. Поскольку метод проб и ошибок (предполагающий произвольный выбор архитектуры и последующее конструирование на ее основе системы под лозунгом «Будь что будет») в качестве оптимального способа выбора архитектуры для системы нас совсем не устраивает, то далее рассмотрим вопросы оценки архитектуры и архитектурного проектирования.

4.1.3.1 Архитектурные образцы, эталонные модели и эталонные варианты архитектуры

В промежутке между схемами из прямоугольников и линий – простейшими «набросками» архитектур – и комплексными архитектурами, укомплектованными всей необходимой информацией о системах, существуют многочисленные переходные этапы. Каждый такой этап есть результат принятия ряда архитектурных решений, совокупность архитектурных альтернатив. Некоторые из них сами по себе имеют определенную ценность. Прежде чем переходить к анализу архитектурных структур, рассмотрим три проме- жуточных этапа.

1. Архитектурный образец – это описание типов элементов и отношений и изложение ряда ограничений на их использование. Образец имеет смысл рассматривать как совокупность ограничений, накладываемых на архитектуру, – конкретнее, на типы элементов и образцы их взаимодействия; на основе этих ограничений складывается ряд или семейство соответствующих им вариантов архитектуры. К примеру, одним из общеупотребительных архитектурных образцов является клиент-сервер. Клиент и сервер – это два типа элементов; их взаимодействие описывается посредством протокола, при помощи которого сервер взаимодействует со всеми своими клиентами. Термин «клиент-сервер» по смыслу лишь предполагает множественность клиентов; конкретные клиенты не перечисляются, и речи о том, какая функциональность, помимо реализации протоколов, характерна для клиентов или для сервера, не идет. Согласно этому (неформальному) определению, образцу «клиент-сервер» соответствует бесчисленное количество различных вариантов архитектуры, причем все они чем-то отличаются друг от друга. Несмотря на то, что архитектурный образец не является архитектурой, он все же содержит весьма полезный образ системы – он накладывает на архитектуру, а следовательно, и на систему, полезные ограничения.

У образцов есть один крайне полезный аспект – дело в том, что они демонстрируют известные атрибуты качества. Именно поэтому архитекторы выбирают образцы не наугад, а исходя из определенных соображений. Некоторые образцы содержат известные решения проблем, связанных с производительностью, другие предназначаются для систем с высокими требованиями к безопасности, третьи успешно реализуются в системах с высокой готовностью. Во многих случаях выбор архитектурного образца оказывается первым существенным решением архитектора.

Синонимичным архитектурному образцу является общеупотребительный термин «архитектурный стиль» (architectural style).

Эталонная модель – это разделение между отдельными блоками функциональных возможностей и потоков данных. Эталонной моделью называется стандартная декомпозиция известной проблемы на части, которые, взаимодействуя, способны ее разрешить. Поскольку эталонные модели имеют происхождение в опыте, их наличие характерно только для сформировавшихся предметных областей. Вы можете назвать стандартные элементы компилятора или системы управления базами данных? А в общих словах объяснить, как эти элементы сообща решают свою общую задачу? Если можете, значит, вы знакомы с эталонной моделью этих приложений.

Эталонная архитектура – это эталонная модель, отображенная на программные элементы (которые сообща реализуют функциональность, определенную в эталонной модели) и потоки данных между ними. В то время как эталонная модель обеспечивает разделение функций, эталонная архитектура отображает эти функции на декомпозицию системы. Соответствие может быть как однозначным, так и неоднозначным. В программном элементе может быть реализована как отдельная часть функции, так и несколько функций сразу.

Эталонные модели, архитектурные образцы и эталонные архитектуры не являются вариантами архитектуры; это не более чем полезные понятия, способствующие фиксации отдельных элементов архитектуры. Каждый из них появляется как результат проектных решений, принимаемых на самых ранних этапах. Отношение между этими проектными элементами представлено на рисунке 4.2.

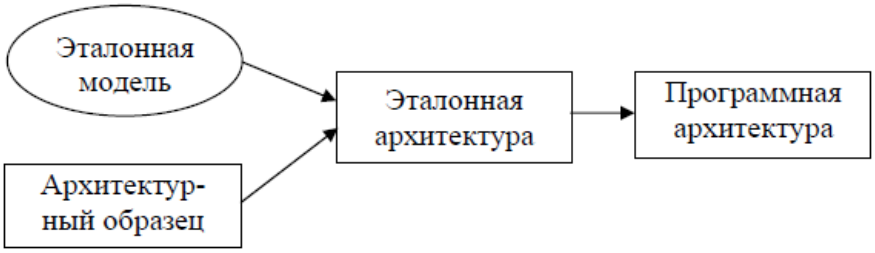


Рисунок 4.2 - Отношения между эталонными моделями, архитектурными образцами, вариантами эталонной и программной архитектуры

Аналогии архитектуры с другими значениями этого слова проводятся весьма часто. Как правило, архитектура ассоциируется с физической структурой (здания, улицы, аппаратура) и физическим расположением. Архитектор, разрабатывающий план здания, имеет целью обеспечить его доступность, эстетическую привлекательность, освещенность, эксплуатационную надежность и др. Программный архитектор в процессе проектирования системы должен стремиться к обеспечению таких характеристик, как параллелизм, модифицируемость, практичность, безопасность и т. д.; кроме того, он призван установить баланс между требованиями к этим характеристикам.

Аналогии между зданиями и программными системами не очень надежны, они слишком быстро оказываются несостоятельными. Хороши они тем, что помогают осознать важность позиции наблюдателя и прийти к выводу о множественности значений понятия «структура» в зависимости от мотивов ее изучения. Точное определение программной архитектуры значительно менее существенно, чем анализ сущности этого понятия.

4.1.3.2 Архитектурные структуры и представления

Современные программные системы настолько сложны, что разбирать их в комплексе крайне сложно. Приходится концентрировать внимание на одной или нескольких структурах программной системы. Для того чтобы рассуждать об архитектуре осмысленно, мы должны определиться с тем, какая структура или какие структуры в данный момент являются предметом обсуждения, о каком представлении (view) архитектуры мы говорим.

Рассматривая представление архитектуры, мы будем употреблять связанные между собой понятия структуры (structure) и представления (view). Представление – это отображение ряда связанных архитектурных элементов в том виде, в котором ими оперируют заинтересованные в системе лица.

В нем фиксируются отображения совокупности элементов и установленных между ними связей. Структура же – это собственно ряд элементов, существующих в рамках программного или аппаратного обеспечения. В частности, модульная структура представляет собой набор модулей системы с указанием их организации. Модульное представление есть отображение этой структуры, документированное и применяемое теми или иными заинтересованными лицами. Несмотря на то, что эти термины иногда используются как синонимы, мы намерены придерживаться приведенных определений.

Архитектурные структуры подразделяются на три общие группы, в каждую из которых включается элементы определенного характера.

Модульные структуры. Элементами таких структур являются модули – блоки реализации.

Модули предполагают рассмотрение системы с точки зрения кода. Им, как отдельным областям, выделяются определенные функциональные обязанности. Особого внимания тому, как конечное программное обеспечение заявит себя в период прогона, в данном случае не уделяется. Модульные структуры позволяют отвечать на такие вопросы, как: Какие основные функциональные обязанности несет данный модуль? К каким программным элементам он может обращаться? Какое программное обеспечение он фактически использует? Между какими модулями установлены отношения обобщения или специализации (например, наследования)?

Структуры «компонент и соединитель». В данном случае элементами являются компоненты (основные единицы вычислений) и соединители (инструменты взаимодействия между компонентами) периода прогона. Среди вопросов, на которые отвечают структуры «компонент и соединитель», – такие, например, как: Каковы основные исполняемые компоненты и как происходит их взаимодействие? Каковы основные совместно используемые хранилища данных? Какие части системы воспроизводятся? Каким образом по системе проходят данные? Какие элементы системы способны исполняться параллельно? Какие структурные изменения происходят с системой во время ее исполнения?

Структуры распределения. Структуры распределения демонстрируют связь между программными элементами, с одной стороны, и элементами одной или нескольких внешних сред, в которых данное программное обеспечение создается и исполняется, – с другой. Они отвечают на вопросы: на каком процессоре исполняется данный программный элемент? в каких файлах каждый элемент хранится в ходе разработки, тестирования и конструирования системы? каким образом программные элементы распределяются между группами разработчиков?

Программные структуры. Наиболее распространенные и полезные программные структуры представлены на рисунке 4.3.

Рисунок 4.3 - Стандартные структуры программной архитектуры

Модуль. Модульные структуры делятся на следующие разновидности.

Декомпозиция. В качестве блоков выступают модули, между которыми установлены отношения «является подмодулем...»; таким образом, крупные модули в рекурсивном порядке разлагаются на меньшие, и процесс этот завершается только тогда, когда мелкие модули становятся вполне понятными. Модули в рамках этой структуры часто используются в качестве отправной точки для последующего проектирования – архитектор перечисляет блоки, с которыми ему предстоит работать, и в расчете на более подробное проектирование, а также, в конечном итоге, на реализацию распределяет их между модулями. У модулей во многих случаях есть связанные продукты (например, спецификации интерфейсов, код, планы тестирования и т. д.). Структура декомпозиции в значительной степени обеспечивает модифицируемость системы – при этом складывается ситуация, когда наиболее вероятные изменения приходятся на долю нескольких небольших модулей. Довольно часто декомпозиция задействуется как основа для организации проекта, включающая структуру документации, планы интеграции и тестирования. Иногда блоки этой структуры получают оригинальные названия (от пользующихся ими организаций). К примеру, в ряде стандартов Министерства обороны США определяются элементы конфигурации компьютерных программ (Computer Software Configuration Items, CSCI) и компоненты компьютерных программ (Computer Software Components, CSC), которые представляют собой не что иное, как блоки модульной декомпозиции.

Варианты использования. Блоками этой важной, но не слишком распространенной структуры могут быть либо модули, либо (в случаях, когда требуется более мелкая структура)

процедуры или ресурсы интерфейсов модулей. Между такими блоками устанавливаются отношения использования (uses relationship). Если для обеспечения правильности первого блока требуется наличие правильной версии (в отличие от заглушки) второго, то последний используется первым. Структура использования полезна при конструировании систем, которые либо легко расширяются дополнительными функциями, либо предполагают возможность быстрого извлечения полезных функциональных подмножеств. Способность без труда разбить рабочую систему на ряд подмножеств подразумевает возможность инкрементной разработки – многофункционального конструкторского приема.

Многоуровневая. Если отношения использования в рамках этой структуры находятся под строгим, особым образом осуществляемым контролем, возникает система уровней – внутренне связанных наборов родственных функций. В рамках строго многоуровневой структуры уровень *n* может обращаться к услугам только в том случае, если они предоставляются уровнем *n*–1. На практике это правило существует в виде многочисленных вариантов (которые частично снимают представленное структурное ограничение). Уровни во многих случаях проектируются в виде абстракций (виртуальных машин), которые, стараясь обеспечить переносимость, скрывают детали реализации нижележащих уровней от вышележащих.

Класс, или обобщение. Блоки модулей в рамках этой структуры называются классами. Отношения между ними строятся по образцам: «наследует от...» и «является экземпляром...». Данное представление способствует анализу коллекций сходного поведения или сходных возможностей (например, классов, которые наследуют от других классов) и параметрических различий, фиксация которых производится путем определения подклассов. Структура классов также позволяет анализировать вопросы повторного использования и инкрементного введения функциональности.

Компонент и соединитель. Среди структур данного вида выделяются следующие:

Процесс или сообщающиеся процессы. Подобно всем прочим структурам «компонент и соединитель», эта является ортогональной по отношению к модульным структурам, а связана она с динамическими аспектами исполняемой системы. В качестве блоков в данном случае выступают процессы или потоки, связь между которыми устанавливается путем передачи данных, синхронизации и/или операций исключения. Здесь, как и во всех остальных структурах «компонент и соединитель», действует отношение прикрепления (attachment), демонстрирующее связь компонентов друг с другом. Структура процессов существенно облегчает конструирование рабочей производительности и готовности системы.

Параллелизм. Данная структура «компонент и соединитель» позволяет архитекторам выявлять перспективы параллелизма и локализовывать возможности состязаний за ресурсы. В качестве блоков выступают компоненты, а соединители играют роль логических потоков. Логическим потоком называется такая последовательность вычислений, которую впоследствии, в ходе процесса проектирования, можно связать с отдельным физическим потоком. Структура параллелизма задействуется на ранних этапах процесса проектирования, способствуя выявлению требований к организации параллельного исполнения.

Совместно используемые данные, или репозиторий. В состав данной структуры входят компоненты и соединители, обеспечивающие создание, хранение и обращение к данным постоянного хранения. Она наилучшим образом приспособлена к таким ситуациям, когда система структурирована на основе одного или нескольких репозиториев совместно используемых данных. Она отражает производство и потребление данных программными элементами периода прогона, а в деле обеспечения высокой производительности и целостности данных эти сведения представляют большую ценность.

Клиент-сервер. Эта структура предназначена для систем, сконструированных в виде группы взаимодействующих клиентов и серверов. В качестве компонентов в данном случае выступают клиенты и серверы, а соединителями являются протоколы и сообщения, которыми они обмениваются в процессе обеспечения работоспособности системы. Такая структура требуется для разделения задач (обеспечивающего модифицируемость), физического распределения и выравнивания нагрузок (обеспечивающего эффективность исполнения).

Распределение. Среди структур распределения выделяются следующие:

Размещение. Структура размещения отражает распределение программного обеспечения между элементами аппаратной обработки и передачи данных. В качестве распределяемых элементов могут выступать программные продукты (как правило, процессы из представления «компонент и соединитель»), аппаратные объекты (процессоры) и каналы передачи

данных. Отношения устанавливаются по распределению и демонстрируют физические устройства, на которых размещаются программные элементы; возможны также отношения миграции, однако они устанавливаются только в случае динамического распределения. Настоящее представление позволяет инженерам анализировать производительность, целостность данных, готовность и безопасность. Все эти характеристики чрезвычайно важны в условиях распределенных и параллельных систем.

Реализация. Данная структура демонстрирует отображение программных элементов (обычно модулей) на файловую структуру (структуры) в условиях разработки системы, интеграции и управления конфигурациями. Это крайне важно в контексте организации разработки и процессов конструирования.

Распределение функций. Данная структура обеспечивает разделение обязанностей по реализации и интеграции модулей между соответствующими группами разработчиков. Наличие в составе архитектуры структуры распределения функций делает очевидным, что при принятии соответствующих решений учитывались как архитектурные, так и организационные факторы. Архитектору должно быть известно, какие навыки требуются от разных групп разработчиков. Кроме того, если речь идет о масштабных распределенных проектах с несколькими источниками, на основе структуры распре- деления функций, можно выявлять функциональные сходства и назначать их одной группе разработчиков, отказываясь, таким образом, от их стихийной многократной реализации.

Общая схема программных структур приводится в таблице 4.1. В ней рассматривается значения элементов, отношения, характерные для каждой из структур, и варианты их практического применения.

Таблица 4.1 - Архитектурные структуры системы

Программная структура	Отношения	Варианты практического применения
Декомпозиция	«Является подмодулем...»; «пользуется скрытой информацией совместно с...»	Распределение ресурсов, структурирование и планирование проекта; информационная закрытость, инкапсуляция; управление конфигурациями
Варианты использования	«Требует наличия...»	Конструирование подмножеств; инженерные расширения
Многоуровневая	«Требует наличия...», «обращается к услугам...», «обобщает...»	Инкрементная разработка; реализация систем на основе переносимости «виртуальных машин»
Классы	«Является экземпляром...», «использует метод доступа из...»	В объектно-ориентированных системах проектирования, производящих на основе универсального шаблона быстрые, почти идентичные реализации
Клиент-сервер	«Обменивается данными с...», «зависит от...»	Распределенное функционирование; разделение задач; анализ производительности; выравнивание нагрузки
Процесс	«Исполняется параллельно с...», «может исполняться параллельно с...», «исключает», «предшествует» и т. д.	Анализ сроков; анализ производительности
Параллелизм	«Исполняется в одном логическом потоке»	Выявление местоположений в которых потоки могут разветвляться, объединяться, создаваться и уничтожаться
Совместно используемые данные	«Производит данные», «потребляет данные»	Производительность; целостность данных, модифицируемость
Размещение	Распределение, миграция	Анализ производительности, готовности и защиты
Реализация	«Хранится в...»	Управление конфигурациями, интеграция, тестирование
Распределение функций	«Назначается...»	Управление проектом, оптимальное использование интеллектуальных ресурсов, управление общностью

Как правило, структура системы анализируется с точки зрения ее функциональности; при этом мы забываем о других ее свойствах: физическом распределении, взаимодействии процессов и синхронизации – все эти свойства обязательно учитываются на уровне архитектуры. Каждая структура содержит метод анализа тех или иных значимых атрибутов качества. К примеру, для того чтобы создать легко расширяемую или сокращаемую систему, необходимо *сконструировать* (именно сконструировать, а не просто зафиксировать) структуру использования. Структура процессов *конструируется* с целью исключения

взаимоблокировки и расширения узких мест. Структура декомпозиции модулей *конструируется* в расчете на производство модифицируемых систем и т. д. Каждая подобная структура снабжает архитектора оригинальным представлением системы и дополнительной базовой точкой проектирования.

4.2 Проектирование архитектуры

4.2.1 Атрибутный метод проектирования

Атрибутный метод проектирования (Attribute-Driven Design, ADD) – это методика определения программной архитектуры, в которой процесс декомпозиции основывается на предполагаемых атрибутах качества продукта. Это рекурсивный процесс декомпозиции, на каждом из этапов которого происходит отбор тактик и архитектурных образцов, удовлетворяющих тем или иным сценариям качества, а также распределение функциональности, направленное на конкретизацию типов модулей данного образца. В контексте жизненного цикла ADD располагается сразу после анализа требований, а начинается он, как мы уже говорили, в тот момент, когда об архитектурных мотивах можно говорить с достаточной степенью уверенности.

Результатом применения ADD являются первые несколько уровней представления декомпозиции модулей архитектуры, а также все другие связанные с ними представления. Не стоит, впрочем, думать, что после ADD становятся известны все детали представлений, – система описывается как набор контейнеров функциональности и существующих между ними взаимосвязей. Во всем процессе проектирования это первый случай сочленения архитектуры, результаты которого, естественно, весьма приблизительны. Тем не менее, в контексте реализации предполагаемых атрибутов качества это очень важный момент, поскольку именно здесь закладываются основы достижения функциональности. Различие между архитектурой, являющейся результатом выполнения ADD, и архитектурой, готовой к реализации, лежит в плоскости принятия подробных проектных решений. В частности, это может быть выбор между конкретными объектно-ориентированными образцами проектирования, с одной стороны, и промежуточным программным обеспечением, применение которого сопряжено с многочисленными архитектурными ограничениями, – с другой. Архитектура, спроектированная средствами ADD, предусматривает отсрочку принятия этого решения, благодаря чему достигается большая гибкость.

С участием общих сценариев, а также тактик и образцов можно создать ряд различных процессов проектирования. Отличаются они принятыми принципами деления проектных работ и содержанием процесса проектирования.

4.2.1.1 Этапы ADD

1. *Выбор модуля для декомпозиции*. Как правило, в качестве исходного модуля берется система в целом. Все необходимые входные данные (ограничения, функциональные требования и требования к качеству) для него должны быть известны.

2. Уточнение модуля в несколько этапов:

а) выбор архитектурных мотивов из набора конкретных сценариев реализации качества и функциональных требований. На этом этапе определяются наиболее важные в контексте проведения декомпозиции моменты;

б) выбор архитектурного образца, соответствующего архитектурным мотивам. Создание (или выбор) образца, тактики которого позволяют реализовать эти мотивы. Выявление дочерних модулей, необходимых для реализации этих тактик;

в) конкретизация модулей, распределение функциональности из элементов Use Case, составление нескольких представлений;

г) определение интерфейсов дочерних модулей. Декомпозиция имеет своим результатом новые модули, а также ограничения на типы взаимодействия между ними. Для каждого из модулей эти сведения следует зафиксировать в документации по интерфейсу;

д) проверка и уточнение элементов Use Case в сценариях качества и наложение, исходя из них, ограничений на дочерние узлы. На этом этапе мы проверяем, все ли факторы учли, а также, в расчете на последующую декомпозицию или реализацию, подготавливаем дочерние модули.

3. Вышеперечисленные этапы следует выполнить в отношении всех нуждающихся в дальнейшей декомпозиции модулей.

4.2.2 Создание макета системы

После проектирования архитектуры и формирования рабочих групп можно приступать к конструированию макета системы. Цель этого этапа в том, чтобы обеспечить основополагающую возможность реализации функциональности системы в предпочтительном (с точки зрения задач проекта) порядке.

Согласно классической методике программной инженерии, следует «заглушать» секции кода так, чтобы различные части системы можно было вводить в действие и тестировать по отдельности. О каких именно частях идет речь? Последовательность реализации всегда можно установить по характеристикам архитектуры.

В первую очередь следует реализовать те программы, которые связаны с исполнением и взаимодействием между архитектурными компонентами.

В системе реального времени это может быть планировщик, в системе на основе правил – процессор правил (с прототипом набора правил), управляющий их активизацией, в многопроцессной системе – механизмы синхронизации процессов, а в клиент-серверной системе – механизм координации действий клиента и сервера. Базовый механизм взаимодействия во многих случаях выражается в виде промежуточных программ стороннего производства; если это так, реализация заменяется установкой. Помимо базовой инфраструктуры передачи данных или взаимодействия имеет смысл ввести в действие ряд простейших функций, инициирующих механическое поведение. На этом этапе в вашем распоряжении уже появляется исполняемая система. Это та основа, на которую можно начинать насаживать дополнительную функциональность.

Теперь выбирайте – какие функциональные элементы следует ввести в эту систему. Решение в данном случае принимается под влиянием нескольких факторов: во-первых, из побуждения снизить риск, обратившись к наиболее трудным областям в первую очередь, во-вторых, исходя из возможностей и специализации имеющегося персонала, и, в-третьих, из соображений «выбросить» продукт на рынок как можно быстрее.

Приняв решение относительно введения в систему очередной порции функциональности, обратитесь к структуре использования – она сообщит о том, какие еще программные средства системы должны корректно исполняться (другими словами, развиться из состояния банальной заглушки в полноценные элементы), для того чтобы реализация выбранных функций стала возможной.

Таким вот образом в систему можно вводить все новые и новые элементы, пока они не закончатся. На любом из таких этапов действия по интеграции и тестированию не представляют большой сложности; равным образом легко обнаружить источник недавно появившихся неисправностей. Чем меньше приращение, тем более предсказуемы бюджет и график и тем больше диапазон решений по поставке у управленцев и специалистов по маркетингу.

Заглушенные элементы, несмотря ни на что, приближают момент завершения работы над системой. Поскольку заглушки относятся к тем же интерфейсам, которые будут присутствовать в окончательной версии системы, даже в отсутствии полноценной функциональности они помогают уяснить и протестировать взаимодействие между компонентами. Проверить это взаимодействие с помощью компонентов-заглушек можно двумя способами: путем генерирования жестко закодированных искусственных выходных данных или считывания таких данных из файла. Кроме того, они способны генерировать искусственную нагрузку, по результатам которой можно приблизительно определить, сколько времени уйдет на фактическую обработку данных в законченной рабочей версии системы. Это помогает на ранней стадии проектирования выявить требования по производительности системы, включая рабочие взаимодействия и узкие места.

Согласно Кусумано (Cusumano) и Селби (Selby), компания Microsoft выстраивает свою стратегию на основе эволюционного жизненного цикла поставки (Evolutionary Delivery Life Cycle). По той версии методики, которой пользуется Microsoft, «завершенный» макет системы создается на раннем этапе жизненного цикла продукта, а впоследствии с некоторой регулярностью (во многих случаях ежедневно), строятся «рабочие» версии с сокращенной функциональностью. В результате получается рабочая система, о достаточности характеристик

которой можно принять решение в любой момент и которую в любой же момент можно развернуть. Есть, впрочем, одна проблема, которую следует предусмотреть, – дело в том, что та рабочая группа, которая заканчивает свою часть системы первой, задает интерфейс, которому должны соответствовать все последующие системы. Это обстоятельство ставит в невыгодное положение наиболее сложные части системы – их разработка сопряжена со значительно большими объемами аналитических действий, вследствие чего вероятность первоочередного определения их интерфейсов сильно снижается. Таким образом, и без этого сложные подсистемы становятся еще более сложными. По нашему убеждению, все согласования по поводу интерфейсов следует проводить на этапе создания макета системы – при таком условии на последующих стадиях разработки можно будет обратить большее внимание на достижение эффективности.

4.3 Документирование программной архитектуры

4.3.1 Варианты применения архитектурной документации

Характер архитектуры любой системы обусловливается предъявляемыми к ней требованиями; это утверждение справедливо и по отношению к документации архитектуры, другими словами, содержание документации зависит от предполагаемых вариантов ее применения. Документация ни при каких обстоятельствах не может быть универсальной. С одной стороны, она должна быть абстрактной и, следовательно, доступной для понимания новыми сотрудниками, но с другой – весьма детальной – настолько, чтобы ее можно было использовать как план проведения анализа. Между архитектурной документацией, предназначенной, скажем, для проведения анализа безопасности, и архитектурной документацией для изучения конструкторами должны существовать серьезные различия. С другой стороны, у этих вариантов будет мало общего с содержанием руководства для ознакомления, предназначенного новому сотруднику.

Архитектурная документация одновременно инструктивна и описательна. Ограничивая диапазон возможных решений, с одной стороны, и перечисляя принятые ранее проектные решения по системе, – с другой, она обязывает соблюдать определенные принципы.

Из всего этого следует, что заинтересованные в составлении документации лица преследуют разные цели – от представленной в ней информации они требуют разной направленности, разных уровней детализации и разных трактовок. Предполагать, что один и тот же архитектурный документ будет одинаково воспринят всеми без исключения заказчиками, наивно. Стремиться следует к тому, чтобы любое заинтересованное лицо смогло быстро найти необходимую информацию, как можно меньше в процессе ее поиска натыкаясь на незначительные (с его точки зрения) сведения.

В некоторых случаях простейшим решением представляется создание нескольких документов, предназначенных для разных заинтересованных лиц. Впрочем, как правило, задача ограничивается созданием единого комплекта документации с дополнительными облегчающими поиск сведений инструкциями.

Согласно одному из фундаментальных правил технической документации в общем и документации программной архитектуры в частности, составитель должен принимать позицию читателя. Без труда составленная, но трудная для восприятия (с точки зрения читателя – в данном случае заинтересованного лица) документация не найдет себе применения.

4.3.2 Представления

Одним из наиболее важных понятий документирования программной архитектуры является «представление» (view). С понятием представления, которое можно кратко определить как способ фиксации структуры, связан основной принцип документирования программной архитектуры.

Документирование архитектуры подразумевает документирование всех значимых представлений с последующей фиксацией сведений, относящихся одновременно к нескольким представлениям.

Принцип этот полезен тем, что он помогает разбить проблему документирования архитектуры на ряд менее обширных элементов:

- выбор значимых представлений;

- документирование представления;
- документирование сведений, относящихся к нескольким представлениям.

4.3.2.1 Выбор значимых представлений

Какие представления следует считать значимыми? Для ответа на этот вопрос необходимо знать заинтересованных лиц и предпочтительные для них способы пользования документацией. Лишь располагая этими сведениями, вы сможете составить удобный для них пакет документации. Любой вариант применения архитектуры как средства постановки задач конструкторов, основы для изучения системы, восстановления ее свойств или планирования проекта можно свести к отдельному заинтересованному лицу, предполагающему задействовать документацию архитектуры соответствующим образом. Не менее серьезное влияние на выбор представлений для дальнейшего документирования оказывают наиценнейшие для большинства заинтересованных в разработке системы лиц атрибуты качества. К примеру, многоуровневое представление (layered view) сообщает сведения о переносимости системы. По представлению размещения (deployment view) можно судить о производительности и надежности системы. И так далее. За отражение этих атрибутов качества в документации ратуют аналитики (а быть может, и сам архи- тектор), в задачу которых входит проверка архитектуры на предмет ее соответствия атрибутам качества.

Различные представления соответствуют отдельным задачам и вариантам использования. Именно по этой причине мы призываем к выбору того или иного представления или набора представлений. Их выбор зависит от задач потребителей документации. Представления ставят акценты на разные элементы системы и/или установленные между ними связи.

Представление отражает произвольный набор элементов системы и установленных между ними отношений. Следовательно, представлением может быть любое сочетание элементов и отношений, которое, по вашему мнению, имеет шансы оказаться интересным части заинтересованных лиц. Ниже приводится состоящая из трех этапов процедура подбора представлений для конкретного проекта.

1. Составьте список возможных представлений. Начните с составления для своего проекта таблицы заинтересованных лиц/представлений. В столбцах выразите применимые к вашей системе представления. Некоторые из них (например, модульное представление и представление вариантов использования) носят универсальный характер; иные (многоуровневое представление, а также большинство представлений из группы «компонент и соединитель», в частности клиент-серверное представление и представление совместно используемых данных) подходят только для соответствующим образом спроектированных систем. Разобравшись с содержанием строк и столбцов, заполните все ячейки, отразив в них степень детализации сведений о каждом представлении, необходимую тем или иным заинтересованным лицам: здесь возможны произвольная детализация, обзор, средняя или высокая детализация.

2. Комбинируйте представления. Скорее всего, количество представлений, отраженных в составленном по результатам первого этапа списке, окажется недопустимым. Для того чтобы сократить список, найдите в таблице представления, требующие не более чем обзорной детализации или служащие ограниченному числу заинтересованных лиц. Проверьте, нельзя ли удовлетворить их запросы другим, более универсальным представлением. Затем найдите представления, которые можно комбинировать — выразить в одном сводном представлении информацию из нескольких исходных представлений. В небольших по масштабу проектах информативное содержание представления реализации, как правило, адекватно отражается в представлении декомпозиции на модули. Последнее, в свою очередь, хорошо сочетается с представлениями вариантов использования и многоуровневым представлением. Наконец, представление размещения можно скомбинировать с любым представителем группы «компонент и соединитель», отражающим распределение компонентов между аппаратными элементами, например с представлением процессов.

3. Расставляйте приоритеты. Представления, оставшиеся после выполнения второго этапа, должны соответствовать потребностям сообщества заинтересованных лиц. Теперь необходимо решить, какие из этих представлений имеют первостепенное значение. Конкретное решение зависит от деталей проекта; впрочем, вы в любом случае должны помнить, что полностью завершать работу над одним представлением, прежде чем приступать к следующему, совершенно не обязательно. Информация, детализированная на уровне обзора, представляет

некоторую ценность, так что наши предпочтения на стороне метода разработки материала «в ширину». Кроме того, интересы одних заинтересованных лиц в ряде случаев ставятся выше интересов других.

4.3.3 Документирование представления

Стандартного шаблона документирования представлений не существу- ет. Поэтому ниже речь пойдет о методике, доказавшей свою жизнеспособ- ность в практической деятельности – стандартной семичастной структуре (seven-part standard organization). Во-первых, вне зависимости от того, каким разделам вы отдадите предпочтение, ввести стандартную структуру совершенно необходимо. Распределение информации по отдельным разделам помогает составителю документации уверенно приступить к решению задачи и установить момент ее выполнения; что же касается читателя, то ему становится проще быстро отыскать интересующую информацию и пропустить все ненужное.

1. Первичное отображение (primary presentation) содержит перечень элементов представления и установленные между ними отношения. В первичном представлении должна содержаться (в форме словаря) та информация о системе, которую необходимо донести до читателя в первую очередь. Это, без сомнения, основные элементы и отношения представления, хотя в некоторых случаях они могут быть отражены не полностью. К примеру, в первичном отображении можно показать элементы и отношения, характерные для нормального режима работы, а сведения об обработке ошибок или исключений представить во вспомогательной документации.

Как правило, первичное отображение составляется в графической форме. Дело в том, что большинство графических нотаций годятся только для первичного отображения – для всех остальных элементов документации они не приспособлены. Неизменным спутником графического представления должен быть перечень условных обозначений, содержащий объяснение использованной нотации и символики или хотя бы указывающий на источники получения этих объяснений.

Иногда первичное отображение составляется в виде таблицы, которая по своему характеру прекрасно подходит для компактного представления больших объемов информации. Требование о кратком выражении наиболее значимых сведений о представлении распространяется и на текст.

2. Каталог элементов (element catalog) предназначен для более детального описания элементов и отношений как участвующих, так и не участвующих в первичном отображении. Архитекторы часто совершают одну и ту же ошибку – уделяя излишне серьезное внимание составлению первичного отображения, они забывают, что без дополнительной информации в нем мало толку. К примеру, если на диаграмме показаны элементы A, B и C, необходимо составить относительно детальное описание сущности этих элементов, их назначения и ролей, причем разместить эту информацию лучше всего в словаре представления. Скажем, для представления декомпозиции на модули характерно наличие элементов-модулей, различных вариантов отношения «является частью», а также свойств, определяющих обязанности каждого модуля. В представлении процессов содержатся элементы-процессы, отношения, определяющие синхронизацию и другие механизмы межпроцессного взаимодействия, а также свойства с временными параметрами.

Кроме того, в каталоге обязательно должны разъясняться все элементы и отношения, значимые для данного представления, но по каким-то причинам не показанные в первичном отображении.

3. На контекстной диаграмме (context diagram) должно быть показано отношение изображенной в представлении системы к своему окружению из словаря представления. К примеру, представление «компонент и соединитель» подразумевает демонстрацию взаимодействия отдельных компонентов и соединителей с внешними компонентами и соединителями через посредство интерфейсов и протоколов.

4. Руководство по изменчивости (variability guide) демонстрирует способы применения изменяемых параметров, входящих в состав показанной в данном представлении архитектуры. В некоторых архитектурах принятие решений откладывается до более поздних стадий процесса разработки, что не отменяет необходимость в составлении документации. Примеры изменчивости обнаруживаются во всех линейках программных продуктов, архитектура которых

предусматривает возможность создания множества конкретных систем. В руководстве по изменчивости следует документировать все изменяемые параметры архитектуры, включая: альтернативы, рассматриваемые при принятии того или иного решения.

В модульном представлении такими альтернативами являются различные варианты параметризации модулей. В представлении «компонент и соединитель» могут быть отражены ограничения по дублированию, планированию или выбору протоколов. В представлении распределения это условия назначения конкретного программного элемента тому или иному процессору; время связывания альтернатив. Одни решения принимаются в период проектирования, другие – в период производства, третьи – в период исполнения.

5. Предпосылки архитектурного решения (architecture background) – это раздел, в котором содержится обоснование отраженного в данном представлении проектного решения. Цель его – объяснить читателю, почему проект выглядит так, как он выглядит, и представить убедительные аргументы его превосходства. В состав этого раздела входят следующие элементы:

- логическое обоснование, демонстрирующее предпосылки принятия проектных решений, отраженных в данном представлении, и причины отказа от альтернатив;
- результаты анализа, оправдывающие проектное решение и объясняющие последствия модификаций;
- отраженные в проектном решении допущения.

6. Глоссарий терминов (glossary of terms), применяемых в представлениях, и краткое описание каждого из них.

7. Другая информация. Содержание этого раздела зависит от методов работы конкретной организации. В частности, здесь можно разместить административные сведения: авторство, данные управления конфигурациями и историю изменений. Кроме того, архитектор волен разместить здесь систему ссылок на отдельные разделы сводки требований. Таким образом, речь идет об информации, которая, строго говоря, не имеет прямого отношения к архитектуре, но которую удобнее всего привести вместе с архитектурными сведениями.

4.3.3.1 Документирование поведения

Представления содержат структурную информацию о системе. Но для рассуждений о некоторых свойствах системы ее недостаточно. К примеру, для того чтобы уяснить вопросы взаимоблокировки, необходимо знать последовательность взаимодействий между элементами, которую структурная информация не отражает. Эти сведения, равно как возможности параллелизма и временные зависимости, характерные для взаимодействий (которые происходят в установленные моменты или по истечении установленных временных периодов), раскрывают поведенческие описания. Поведение документируется применительно к отдельному элементу или к множеству взаи- модействующих элементов. Выбор в вопросах моделирования зависит от типа проектируемой системы. К примеру, если речь идет о встроенной системе реального времени, на первое место выходят свойства времени и время наступления событий. В системе банковского обслуживания значительно важнее фактического времени наступления событий оказывается их последовательность (например, последовательность элементарных транзакций и процедур отката). В зависимости от вида предполагаемых аналитических действий уместно обращаться к разным методикам моделирования и нотациям. Примерами поведенческих описаний в языке UML являются диаграммы последовательностей и схемы состояний. Подобные нотации весьма широко распространены.

Схемы состояний как формализм появились в 1980-х гг. и изначально предназначались для описания реактивных систем. Ряд реализованных в них полезных расширений дополняет традиционные диаграммы состояний (такие, как вложенность состояний и состояния «и») и придает выразительность абстракции и параллелизму моделей. Схемы состояний позволяют рассуждать о системе в целом. Предполагается отображение всех ее состояний, а методики анализа в отношении системы приобретают универсальный характер. Становятся возможными ответы на вопрос типа: всегда ли время отклика на данный стимул будет меньше 0,5 секунды?

Диаграмма последовательностей помогает документировать последовательность обменов стимулами. Она отражает кооперацию применительно к экземплярам компонентов и их взаимодействию, причем последнее представляется во временной последовательности. Вертикальное измерение при этом выражает время, а горизонтальное – различные компоненты. Диаграммы последовательностей позволяют строить умозаключения на основе конкретных

сценариев использования. Они демонстрируют механизм реагирования системы на отдельные стимулы, иллюстрируют выбор путей в системе и отвечают на вопрос типа: какие параллельные операции приходят в момент реагирования системы на определенные стимулы в определенных условиях?

4.3.3.2 Документирование интерфейсов

Интерфейс (interface) – это граница, на которой встречаются, взаимодействуют или передают друг другу информацию две независимые сущности.

Очевидно, что интерфейсы элементов – носителей их внешне видимых другим элементам свойств – являются понятием архитектурным. Поскольку без них невозможны ни анализ, ни проектирование систем, документировать интерфейсы совершенно необходимо.

Под документированием интерфейса подразумевается указание его имени, идентификация, а также отражение всех синтаксических и семантических сведений о нем. Первые два элемента – указание имени и идентификация – обобщенно называются «сигнатура» интерфейса. Если в качестве ресурсов интерфейса выступают вызываемые программы, сигнатура обозначает их и определяет их параметры. Параметры определяются по порядку, типу данных и (иногда) по принципу возможности изменения их значений программой. Та информация о программе, которая содержится в сигнатуре, обычно приводится в заголовочных файлах C или C++ и в интерфейсах Java.

При всей полезности сигнатур (в частности, они делают возможной автоматическую проверку конструкций) ими все не исчерпывается. Соответствие сигнатур обеспечивает успешную компиляцию и/или компоновку системы, но совершенно не гарантирует достижение конечной цели – ее нормальное функционирование. Необходимая для этого информация относится к семантике интерфейса, которая сообщает, что происходит при активизации ресурсов.

Интерфейс документируется в форме спецификации – изложения свойств элемента, которые архитектор желает предать огласке. Это должна быть только та информация, которая необходима для организации взаимодействия с интерфейсом. Другими словами, архитектор должен решить, во-первых, какую информацию об элементе допустимо и уместно сообщить читателю и, во-вторых, какая информация, вероятнее всего, не будет подвержена изменениям. В процессе документирования интерфейса важно, с одной стороны, не раскрыть слишком много сведений, и с другой – не утаить необходимые данные. Разработчики не смогут наладить успешное взаимодействие с элементом, если о нем будет недостаточно информации. Избыток информации, в свою очередь, усложняет будущие изменения в системе и усиливает их влияние, делает интерфейс неудобным для восприятия. Для того чтобы достойно справиться с этой ситуацией, наибольшее внимание следует уделять не реализации элементов, а их взаимодействию с рабочими средами. Документированию подлежат только внешне видимые явления.

Элементы, присутствующие в виде модулей, часто напрямую соответствуют одному или нескольким элементам представления «компонент и соединитель». Как правило, интерфейсы элементов в представлении модулей и представлении «компонент и соединитель» схожи или идентичны и документировать их в обоих этих представлениях излишне. Поэтому спецификацию интерфейса следует привести в модульном представлении, а в «компоненте и соединителе» поместить ссылку на нее и изложить индивидуальную для данного представления информацию. Кроме того, один модуль может оказаться общим для нескольких модульных представлений – например, декомпозиции на модули и использования. В таком случае, как и в предыдущем, спецификацию интерфейса следует привести в одном из этих представлений, а во всех остальных поставить соответствующую ссылку.

Шаблон для документирования интерфейсов. Ниже изложен один из вариантов стандартной структуры документации интерфейса. Некоторые ее элементы, если они оказываются бесполезными в конкретном применении, можно выкинуть, другие, наоборот, ввести. Значительно более важными, чем сама стандартная структура, представляются нам способы ее применения. Ваша цель должна заключаться в том, чтобы в точности изложить все внешне видимые взаимодействия интерфейсов, предусмотренных в проекте.

Индивидуальность интерфейса. Если у элемента несколько интерфейсов, их нужно как-то отличать друг от друга. Как правило, для этого интерфейсам присваиваются имена, в некоторых случаях сопровождающиеся указанием номера версии.

Предоставляемые ресурсы. Основным предметом документирования любого интерфейса должны быть предоставляемые им ресурсы. Они определяются синтаксисом, семантикой (описывающей следствия их применения) и ограничениями по использованию. Существует ряд нотаций, предназначенных для документирования синтаксиса интерфейса. Одна из них – язык описания интерфейсов (interface definition language, IDL), разработанный рабочей группой OMG, – применяется в сообществе CORBA. С помощью специальных языковых конструкций этого языка описываются типы данных, операции, атрибуты и исключения. Семантическую информацию можно выразить только в комментариях. В большинстве программных языков есть встроенные средства спецификации сигнатур элементов, примером чему – заголовочные файлы (.h) в C и спецификации пакетов в Ada. Наконец, на языке UML синтаксическая информация об интерфейсе выражается через стереотип interface. Для интерфейса необходимо хотя бы ввести имя; кроме того, архитектор волен составить для него сигнатуру.

Синтаксис ресурса. Здесь указывается сигнатура ресурса. Содержащейся в сигнатуре информации должно быть достаточно для того, чтобы любая сторонняя программа смогла корректно с точки зрения синтаксиса обратиться к программе, использующей данный ресурс. Таким образом, в сигнатуру входят имя ресурса, имена и логические типы данных его аргументов (если таковые имеются) и т. д.

Семантика ресурса. Здесь приводится описание результатов вызова ресурса, в частности:

- присваивание значений данным, к которым может обращаться актер, вызывающий рассматриваемый ресурс – от простого задания значения возвращаемого аргумента до обновления центральной базы данных;
- события и сообщения, сигнализируемые/отправляемые в результате обращения к ресурсу;
- предполагаемое поведение других ресурсов как результат обращения к рассматриваемому ресурсу (к примеру, если данный ресурс уничтожит некий объект, то последующие попытки обращения к этому объекту будут приводить к новому результату – ошибке);
- результаты, наблюдаемые пользователем (встречающиеся в основном во встроенных системах: скажем, вызов программы, ответственной за включение бортового индикатора, приводит к наблюдаемому результату).

Кроме того, семантика должна прояснять вопросы, связанные с исполнением ресурса: будет ли он носить элементарный характер, можно ли его приостановить или прервать. Как правило, семантическая информация выражается средствами естественного языка. Для фиксации предусловий и постусловий – сравнительно простого и эффективного метода выражения семантики – специалисты во многих случаях прибегают к булевой алгебре. Еще одним средством передачи семантической информации являются следы – запи- си последовательностей операций и взаимодействий, описывающих реакцию элемента на тот или иной вариант использования.

Ограничения на использование ресурсов. В каких обстоятельствах возможно обращение к рассматриваемому ресурсу? Существует ли необходимость в предваряющей его считывание инициализации данных? Обусловливается ли вызов одного метода вызовом другого? Не установлены ли какието ограничения относительно количества актеров, имеющих возможность одномоментного взаимодействия с ресурсом? Возможно, в силу принадлежности элемента определенному актеру, он единственный, кто обладает пол- номочиями по модификации элемента, а все остальные актеры ограничиваются лишь считыванием. Не исключено также, что, согласно многоуровневой схеме безопасности, каждый актер может обращаться к строго определенным ресурсам или интерфейсам. Если рассматриваемый ресурс отталкивается от каких-либо допущений в своем окружении (например, требует наличия других ресурсов), их следует задокументировать.

Определения типов данных. Если какой-либо ресурс интерфейса задействует тип данных, отличный от предусмотренного соответствующим языком программирования, архитектор должен привести определение типа данных. Если он определен в другом элементе, достаточно установить ссылку на его документацию. Как бы то ни было, программисты, которые пишут элементы, обращаясь к такому ресурсу, должны знать:

а) как объявлять переменные и константы типа данных;

б) как в рамках этого типа данных записывать литеральные значения;

в) какие операции и сравнения применимы к членам типа данных;

г) как преобразовывать значения этого типа данных в данные других типов.

Определения исключений. Здесь предполагаются описания исключений, которые могут порождать ресурсы на данном интерфейсе. Поскольку одни и те же исключения во многих случаях характерны для множества ресурсов, имеет смысл отделять список исключений каждого ресурса от их определений, причем последние размещать в специальном словаре. Настоящий раздел как раз исполняет роль словаря. Здесь же допустимо определять стандартное поведение обработки ошибок.

Характерная для интерфейса изменчивость. Предполагает ли данный интерфейс возможность конфигурирования элемента? Подобные параметры конфигурации (configuration parameters), а также их воздействие на семантику интерфейса подлежат документированию. В качестве примеров изменчивости можно привести емкость видимых структур данных и рабочие характеристики соответствующих алгоритмов. Для каждого параметра конфигурации архитектор должен зафиксировать имя, диапазон значений и время связывания его фактических значений.

Характеристики атрибутов качества интерфейса. Архитектор должен задокументировать характеристики атрибутов качества (например, производительность или надежность), предоставляемых интерфейсом конечным пользователям. Эти сведения можно выразить в форме ограничений на реализацию составляющих интерфейс элементов. Выбор атрибутов качества, на которых предполагается сконцентрироваться и принять обязательства, проводится в зависимости от контекста.

Требования к элементам. Среди требований элемента может значиться наличие конкретных именованных ресурсов других элементов. Документация в данном случае составляется так же, как и в отношении ресурсов, – указываются синтаксис, семантика и ограничения по использованию. В некоторых случаях подобного рода информацию удобнее документировать в форме ряда допущений о системе, принятых проектировщиком элемента.

В таком случае требования можно представить на суд экспертов, которые уже на ранних стадиях процесса проектирования смогут подтвердить или опровергнуть принятые допущения.

Логическое обоснование и соображения о проекте. Как и в случае с логическим обоснованием архитектуры в целом (или архитектурных представлений), архитектор должен документировать факторы, обусловившие принятие тех или иных проектных решений относительно интерфейса. В логическом обосновании необходимо указать мотивацию принятия этих решений, ограничения и компромиссы, обрисовать рассмотренные, но впоследствии забракованные решения (не забыв попутно изложить причины отказа от них), и изложить соображения архитектора касательно дальнейших изменений интерфейса.

Руководство по использованию. В некоторых случаях требуется анализ семантики с позиции связей между множеством отдельных взаимодействий.

Если это так, в дело вступает протокол (protocol), документировать который следует согласно последовательности взаимодействий. Протоколы отражают комплексное поведение при взаимодействии тех образцов использования, которые, по мысли архитектора, должны встречаться с определенной регулярностью. Сложность взаимодействия с элементом через его интерфейс следует компенсировать статической поведенческой моделью (например, схемой состояний) или примерами проведения отдельных видов взаимодействия (в форме диаграмм последовательностей).

4.4 Методы анализа архитектуры

4.4.1 Метод анализа компромиссных архитектурных решений – комплексный подход к оценке архитектуры

Метод анализа компромиссных архитектурных решений (Architecture Tradeoff Analysis Method, ATAM) – комплексная универсальная методика оценки программной архитектуры. В соответствии с названием этот метод обнаруживает степень реализации в архитектуре тех или иных задач по качеству, а также (исходя из допущения о том, что любое архитектурное решение влияет сразу на несколько задач по качеству) механизм их взаимодействия – другими словами, их взаимозаменяемость.

Основное назначение ATAM состоит в том, чтобы выявить коммерческие задачи, поставленные в контексте разработки системы и проектирования архитектуры. Вкупе с

участием заинтересованных лиц это помогает специалистам по оценке сфокусироваться на тех элементах архитектуры, которые играют первостепенную роль для реализации упомянутых задач.

4.4.1.1 Этапы ATAM

Операции оценки по методу ATAM распадаются на четыре этапа.

На нулевом этапе – «Установление партнерских отношений и подготовка» – руководители группы оценки проводят неофициальные совещания с основными ответственными за проект лицами и прорабатывают подробности предстоящей работы. Представители проекта посвящают специалистов по оценке в суть проекта, тем самым повышая квалификацию некоторых из них. Эти две группы принимают согласованные логистические решения: где и когда встречаться, кто предоставит лекционные плакаты и т. д. Кроме того, они согласовывают предварительный перечень заинтересованных лиц (перечисляя их не по именам, а по ролям), устанавливают сроки и получателей сводного отчета. Они также организуют снабжение специалистов по оценке архитектурной документацией – если, конечно, таковая существует и может оказаться полезной. Наконец, руководитель группы оценки объясняет руководителю проекта и архитектору, какую информацию им следует предоставить на первом этапе, и при необходимости помогает им составить соответствующие презентации.

На первом и втором этапах проводится непосредственно оценка – все погружены в аналитические операции. К началу этих этапов участники группы оценки должны ознакомиться с документацией по архитектуре, получить достаточное представление о системе, знать задействованные архитектурные методики и ориентироваться в первостепенных атрибутах качества. На первом этапе, намереваясь приступить к сбору и анализу информации, участники группы оценки встречаются с лицами, ответственными за проект (как правило, встреча длится весь день). На втором этапе к специалистам присоединяются заинтересованные в архитектуре лица, и в течение примерно двух дней они проводят аналитические мероприятия совместно.

Третий этап занимает доработка – группа оценки составляет в письменном виде и предоставляет получателям сводный отчет. По сути, участники занимаются самопроверкой и вносят в результаты своей работы разного рода коррективы. На заключительном совещании группы обсуждаются успехи и трудности. Участники изучают отчеты, выданные им на первом и втором этапах, и заслушивают выступление наблюдателя за процессом. По сути, они занимаются поиском путей усовершенствования по части исполнения своих ролей, с тем чтобы проводить последующие оценки с меньшими усилиями и с более высокой эффективностью. Действия, выполненные в период оценки участниками трех групп, тщательно регистрируются. По прошествии нескольких месяцев руководитель группы должен связаться с заказчиком оценки, для того чтобы оценить долгосрочные результаты ее проведения и сравнить издержки с выгодами.

Четыре этапа ATAM, их участники и приблизительный график представлены в таблице 4.2.

Таблица 4.2 - Этапы ATAM и их характеристики

Этап	Операции	Участники	Средняя продолжительность
0	Установление партнерских отношений и подготовка	Руководство группы оценки и основные ответственные за проект лица	Проходит в неформальной обстановке, согласно конкретной ситуации; может длиться несколько недель
1	Оценка	Группа оценки и ответственные за проект лица	1 день с последующим перерывом продолжительностью от 2 до 3 недель
2	Оценка (продолжение)	Группа оценки, ответственные за проект лица и заинтересованные лица	2 дня
3	Доработка	Группа оценки и заказчик оценки	1 неделя

4.4.2 Метод анализа стоимости и эффективности – количественный подход к принятию архитектурно-проектных решений

Метод анализа стоимости и эффективности (Cost Benefit Analysis Method, CBAM) базируется на методе ATAM и обеспечивает моделирование затрат и выгод, связанных с принятием архитектурно-проектных решений, и способствует их оптимизации. Методом CBAM оцениваются технологические и экономические факторы, а также сами архитектурные решения.

4.4.2.1. Контекст принятия решений

Все программные архитекторы и ответственные лица стремятся довести до максимума разницу между выгодами, полученными от системы, и стоимостью реализации ее проектного решения. Являясь логическим продолжением метода ATAM, CBAM основывается на его артефактах. Контекст CBAM изображен на рисунке 4.4.

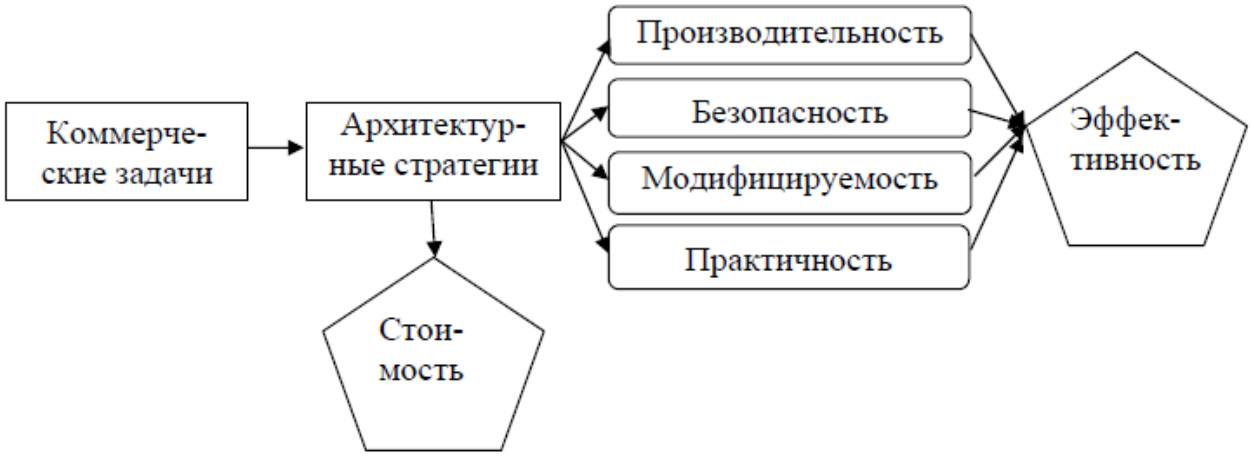


Рисунок 4.4 - Контекст метода анализа стоимости и эффективности (CBAM)

Поскольку архитектурные стратегии ограничиваются разнообразными техническими и экономическими факторами, стратегии, применяемые программными архитекторами и проектировщиками, должны быть поставлены в зависимость от коммерческих задач программной системы. Прямой экономический фактор – это стоимость реализации системы. Техническими факторами являются характеристики системы – другими словами, атрибуты качества. У атрибутов качества также есть экономический аспект – выгоды, получаемые от их реализации.

Как вы помните, по результатам оценки программной системы по методу ATAM в нашем распоряжении оказался ряд документированных артефактов.

Описание коммерческих задач, определяющих успешность системы.

Набор архитектурных представлений, документирующих существующую или предложенную архитектуру.

Дерево полезности, выражающее декомпозицию задач, которые заинтересованные лица ставят перед архитектурой, — от обобщенных формулировок атрибутов качества до конкретных сценариев.

- Ряд выявленных рисков.
- Ряд точек чувствительности (архитектурных решений, которые оказывают влияние на отдельный показатель атрибута качества).
- Ряд точек компромиссов (архитектурных решений, которые воздействуют сразу на несколько показателей атрибута качества, причем на одни положительно, а на другие отрицательно).

ATAM помогает выявить ряд основных архитектурных решений, значимых в контексте сформулированных заинтересованными лицами сценариев атрибутов качества. Эти решения приводят к реакции со стороны атрибутов качества, точнее говоря, отдельных уровней

готовности, производительности, безопасности, практичности, модифицируемости и т. д. С другой стороны, каждое архитектурное решение связано с определенными издержками (стоимостью). К примеру, достижение желаемого уровня готовности путем резервирования аппаратные средств подразумевает один вид издержек, а регистрация в файлах на диске контрольных точек – другой. Эти архитектурные решения приводят к (предположительно разным) измеримым уровням готовности, имеющим определенную ценность для компании-разработчика системы. Возможно, ее руководство полагает, что заинтересованные лица заплатят большую сумму за систему с высокой готовностью (если, к примеру, это телефонный коммутатор или программное обеспечение для медицинского наблюдения), или боится погрязнуть в судебных разбирательствах в случае отказа системы (вполне разумно, если речь идет о программе управления антиблокировочной тормозной системой автомобиля).

ATAM обнаруживает архитектурные решения, принятые относительно рассматриваемой системы, и устанавливает их связь с коммерческими задачами и количественной мерой реакции атрибутов качества. Принимая эти данные на вооружение, CBAM помогает выявить связанные с такими решениями издержку и выгоды. Основываясь на этой информации, заинтересованные лица могут принять окончательные решения относительно резервирования аппаратной части введения контрольных точек и всех прочих тактик, направленных на повышение готовности системы. Вполне возможно, что они предпочтут сконцентрировать ресурсы, которые, как известно, ограниченны, на реализацию какого-то другого атрибута качества – например, на улучшение соотношения выгод и издержек за счет повышения производительности. Из-за ограниченности бюджета разработки и обновления системы каждое архитектурное решение по большому счету соревнуется за право на существование со всеми остальными.

Подобно финансовому консультанту, который никогда напрямую не укажет, во что вкладывать деньги, CBAM не заменяет собой решений, принимаемых заинтересованными лицами. Он лишь помогает им установить и документировать издержки и выгоды архитектурных инвестиций, осознать неопределенность этого «портфеля». На этой основе заинтересованные лица могут принимать рациональные решения, удовлетворяющие их потребности и сводящие к минимуму риски.

Метод CBAM исходит из предположения о том, что архитектурные стратегии (как совокупность архитектурных тактик) оказывают влияние на атрибуты качества системы, а те, в свою очередь, предоставляют заинтересованным лицам некоторые выгоды. Эти выгоды мы называем полезностью (utility). Любая архитектурная стратегия отличается той или иной полезностью для заинтересованных лиц. С другой стороны, есть издержки (стоимость) и время, которые необходимо потратить на реализацию этой стратегии. Отталкиваясь от этой информации, метод CBAM помогает заинтересованным лицам в процессе выбора архитектурных стратегий, характеризующихся максимальной прибылью на инвестированный капитал (return on investment, ROI), – другими словами, наиболее выгодных с точки зрения соотношения выгод и издержек.

4.4.2.2 Реализация CBAM

На рисуноке 4.5 изображена диаграмма процессов, составляющих основу CBAM. Первые четыре этапа на ней сопровождаются комментариями с указанием относительного числа рассматриваемых сценариев. Постепенно их число уменьшается, таким образом, заинтересованные лица сосредоточиваются на тех сценариях, которые, по их мнению, в контексте ROI возымеют наибольшее значение.

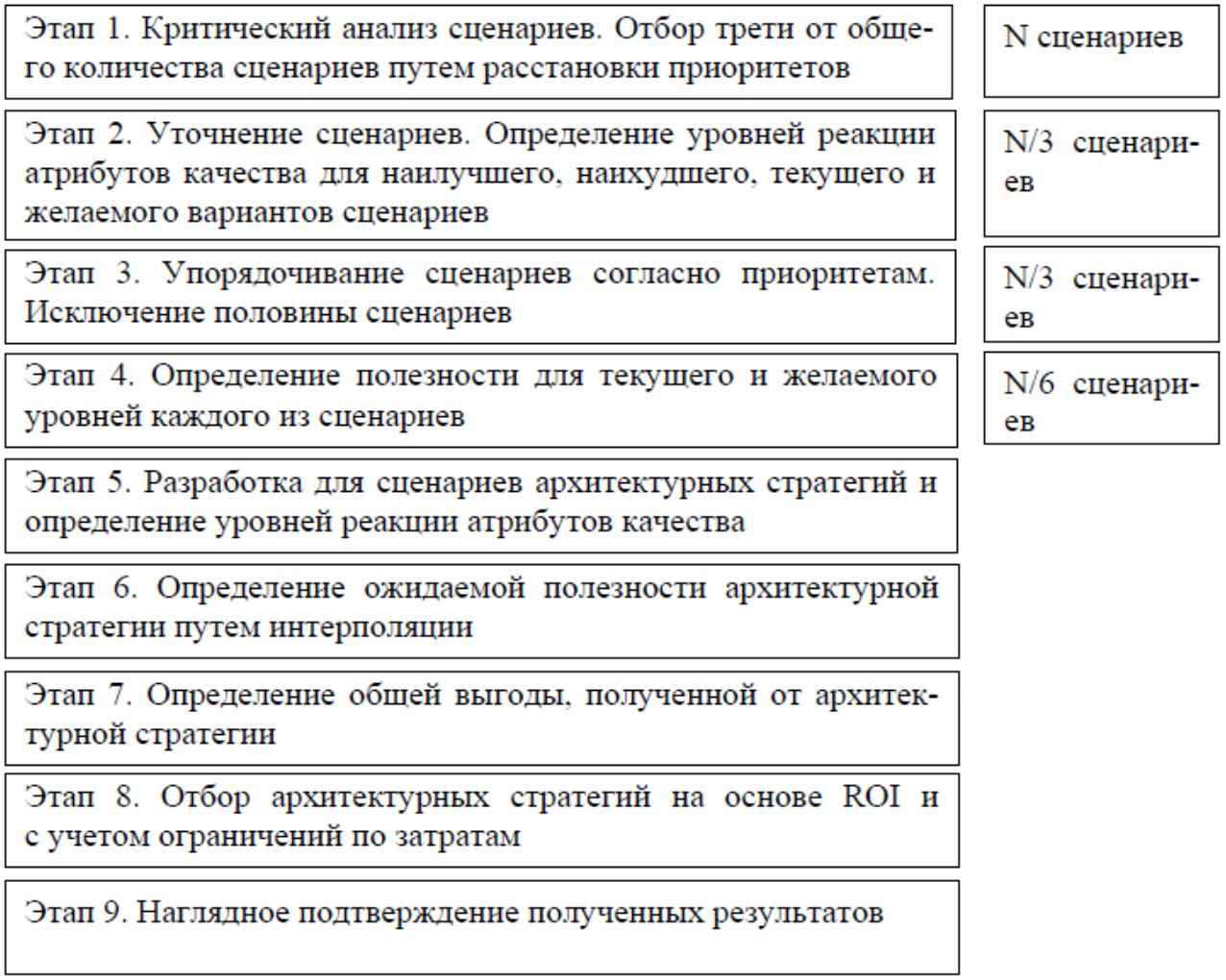


Рисунок 4.5 - Диаграмма процессов CBAM

Этап 1. Критический анализ сценариев. Критический анализ сценариев проводится в рамках ATAM; на этом же этапе заинтересованные лица могут формулировать новые сценарии. Приоритеты расставляются в соответствии с потенциалом сценариев в контексте выполнения коммерческих задач системы; по результатам этапа для дальнейшего рассмотрения отбирается треть от общего первоначального числа сценариев.

Этап 2. Уточнение сценариев. Уточнению подвергаются сценарии, отобранные по результатам первого этапа; основное внимание при этом уделяется их значениям стимула-реакции. Для каждого сценария устанавливаются наихудший, текущий, желаемый и наилучший уровни реакции атрибута качества.

Этап 3. Расстановка сценариев согласно приоритетам. Каждому заинтересованному лицу выделяется 100 голосов, которые он берется распределить между сценариями, исходя из их желаемых значений реакции. После подсчета голосов для дальнейшего анализа остается только половина сценариев. Сценарию с наивысшим рангом присваивается вес 1.0, и, отталкиваясь от него, значения веса устанавливаются для всех остальных сценариев.

Именно эти значения впоследствии задействуются при вычислении общей выгоды стратегии. Помимо прочего, на рассматриваемом этапе составляется список атрибутов качества, которые заинтересованные лица считают значимыми.

Этап 4. Установление полезности. Для сценариев, оставшихся после проведения этапа 3, определяются значения полезности всех уровней реакции (наихудшего, текущего, желаемого, наилучшего) атрибута качества.

Этап 5. Разработка для сценариев архитектурных стратегий и установление их желаемых уровней реакции атрибута качества. Разработка (или фиксация разработанных) архитектурных стратегий, ориентированных на реализацию выбранных сценариев, и определение ожидаемых уровней реакции атрибута качества. Учитывая то обстоятельство, что одна архитектурная стратегия иногда оказывает воздействие на несколько сценариев, расчеты

необходимо провести для каждого из затронутых сценариев.

Этап 6. Определение полезности ожидаемых реактивных уровней атрибута качества путем интерполяции. Исходя из установленных значений полезности (отраженных на кривой полезности) для рассматриваемой архитектурной стратегии определяется полезность желаемого уровня реакции атрибута качества. Эта операция проводится для каждого из перечисленных на этапе 3 значимых атрибутов качества.

Этап 7. Расчет общей выгоды, полученной от архитектурной стратегии. Значение полезности «текущего» уровня вычитается из желаемого уровня и нормализуется исходя из поданных на третьем этапе голосов. Суммируются выгоды, полученные от конкретной архитектурной стратегии, по всем сценариям и для всех значимых атрибутов качества.

Этап 8. Отбор архитектурных стратегий с учетом ROI, а также ограничений по стоимости и времени. Для каждой архитектурной стратегии определяются стоимостные и временные факторы. Значение ROI для стратегий определяется как отношение выгоды к издержкам. Архитектурные стратегии упорядочиваются по рангу согласно значениям ROI; впоследствии бюджет в первую очередь расходуется на высшие по рангу стратегии.

Этап 9. Интуитивное подтверждение результатов. Проверяется соответствие выбранных архитектурных стратегий коммерческим задачам компании. Если наблюдаются противоречия, ищем недосмотры во время проведения анализа. В случае если противоречия значительны, перечисленные этапы проводятся повторно.

План конспекта:

1. Планирование архитектуры.
2. Проектирование архитектуры.
3. Документирование программной архитектуры.
4. Методы анализа архитектуры.

Работа с литературой:

Рекомендуемые источники информации (№ источника)			
Основная	Дополнительная	Методическая	Интернет-ресурсы
1-3	1-3	1-3	1-2

Тема самостоятельного изучения № 5
Технология MDA.

Вид деятельности студентов: самостоятельное изучение литературы
Итоговый продукт самостоятельной работы: конспект
Средства и технологии оценки: собеседование
Краткие теоретические сведения

5 ТЕХНОЛОГИЯ MDA

5.1 Использование архитектуры,управляемой моделью

5.1.1 Концепция архитектуры, управляемой моделью

Технология Model Driven Architecture (MDA) – архитектура, управляемая моделью, была разработана независимой некоммерческой организацией Object Management Group (OMG) – консорциум объектного управления. Она объединяет сотни компаний-производителей программного и аппаратного обеспечения.

В технологии MDA основным элементом проектирования считается модель.

Модель – это описание реальной системы, учитывающее определенные характеристики или аспекты моделируемого объекта, процесса или явления.

В конкретном проекте модель MDA представляет собой законченное и формализованное описание создаваемого программного продукта или его смысловой части. Высокая точность и непротиворечивость описания необходима для автоматизации процесса перевода модели в программный (обычно исходный) код.

Понятие «управление моделью» подразумевает прямое использование модели при любых действиях по проектированию, разработке и развертыванию системы. При внесении изменений в архитектуру приложения модель считается первичной. Пусть, например, требуется пополнить описание класса новым нулем или методом. В классическом (немодельном) подходе к разработке модификация описания класса выполнялась бы в исходном коде, ручным кодированием. В технологии MDA происходит модификация визуальной диаграммы, на которой класс представлен в виде графического элемента. На базе такой диаграммы исходный код с измененным описанием класса генерируется автоматически.

Под архитектурой, управляемой моделью, понимается полная и законченная архитектура приложения, целиком формируемая с помощью модели.

В жизненный цикл программы входит этап формализации и анализа требований заказчика. Идея ведения этого этапа с помощью визуальных моделей развивается уже много лет. Она реализована в виде визуальных высокоуровневых средств, понятных людям, слабо знакомым с технологиями программирования. Такие средства задают целостную внутреннюю архитектуру сложной информационной системы. Лучшие подобные решения поддерживают прямую и двустороннюю связь модели и программного кода. Из модели можно автоматически получить исходный код, а из исходного кода – визуальную модель. В результате удается плавно состыковать этап выработки и согласования требований с этапом кодирования и формирования исполнимого приложения.

В крупных проектах с моделью обычно работает не программист, а системный аналитик. Он отвлекается от мелких деталей реализации и перестает мыслить в терминах отдельных операторов языка программирования. Хороший проектировщик способен охватить и продумать структуру больших функциональных логических блоков и основные взаимосвязи между ними. Далее подобная модель детализируется и транслируется в исходный код на выбранном языке.

Архитектура (приложения), управляемая моделью, позволяет быстро создавать объемные программные системы разной функциональной направленности. У заказчика появляется возможность инвестировать основные средства не в конкретные технические решения, а, прежде всего, в реализацию требуемой логики приложения на модельном уровне, которая потом может почти автоматически развертываться на разных платформах. При этом достигается высокий уровень сохранности инвестиций. Ведь систему, созданную с помощью MDA, при появлении новых технологий можно адаптировать к ним с минимальными усилиями модификации модели и на высоком абстрактном уровне.

Технология MDA ориентирована на создание приложений, которые независимы от платформы, операционной системы и языков программирования. Она позволяет строить масштабируемые приложения из компонентов, которые могут использоваться повторно и многократно. Сам процесс разработки выполняется, как явствует из названия, под управлением модели.

Модель приложения – это взаимосвязанный набор визуальных диаграмм, наглядно описывающих внутреннюю структуру системы и принципы ее функционирования. Модель приложения не привязана к конкретному языку или конкретной среде программирования.

Визуальные диаграммы чаще всего строятся с помощью унифицированного языка моделирования UML (Unified Modeling Language). Его стандарт разработан группой OMG для задач объектно-ориентированного проектирования.

5.1.2 Модельные точки зрения и модели MDA

Когда разработчик взаимодействует с создаваемой системой в рамках подхода MDA, ему доступны три модельные точки зрения на систему.

Точка зрения, независимая от вычислительных особенностей Computation Independent Viewpoint (CIV), нацелена на анализ системной среды и системных требований. Логика работы пока что не рассматривается, возможно, она еще не определена. В CIV учитываются программные, аппаратные и другие технологические ограничения, которые не связаны напрямую с внутренней архитектурой разрабатываемой программы.

Точка зрения, независимая от программно-аппаратной платформы Platform Independent Viewpoint (PIV), задает конкретные функциональные элементы системы, не зависящие от платформы. В ее рамках используются языки моделирования наподобие UML.

Точка зрения, зависящая от платформы Platform Specific Viewpoint (PSV), задает детали реализации, зависящие от используемых платформ и языков программирования. Она сочетает платформенно-независимую логику с дополнительными особенностями, вызванными использованием конкрет- ной платформы или системы.

Каждая из этих точек зрения предлагает разные средства построения соответствующих моделей. Бизнес-аналитики, незнакомые с особенностями программной разработки, вырабатывают свои требования к системе на уровне CIV – с помощью так называемых вычислительно-независимых моделей (Computation Independent Model, CIM). Модели CIM задают основные требования к проекту и включают словари предметной области. Никакие технические характеристики в этих моделях не фиксируются.

Методология MDA описывает не столько моделирование, сколько метамоделирование, предусматривающее большую гибкость в конкретных, прикладных подходах к моделированию.

Метамоделирование – способ описания моделей, определяющий механизмы построения конкретных моделей программных систем с помощью базового словаря и набора ограничений, налагаемых на создаваемые модели. Сегодня вместо термина BMDA корпорация Borland применяет новый термин ECO. Технология ECO (Enterprise Core Objects) – ключевые корпоративные объекты, – реализующая концепцию MDA, стала наиболее важным улучшением последних версий среды Delphi. Она представлена в Delphi 2006 в виде третьей версии ECO III. Каждый компонент ECO представляет собой своеобразную программную «обертку» положений концепции MDA. Он является промежуточным слоем между средствами визуального проектирования программных моделей и их конкретной реализацией на языках программирования Delphi и C+.

Практика перевода требований заказчика напрямую в программный код на языке программирования почти всегда страдает неполноценностью. Заказчик мыслит одними понятиями, программист – совсем другими. Поэтому они не подозревают о множестве потенциально возможных проблем. Так, заказчику какие-то моменты автоматизации выбранных процессов могут казаться очевидными. Он о них и не упоминает, считая, что программа выполнит определенный набор действий сама. Программист, наоборот, подходит к созданию системы с узких технических позиций, программируя только очевидный набор функций, заданный в техническом задании, и редко учитывает все множество взаимосвязей между большим числом требований. В результате в середине проекта выясняется, что архитектуру решения невозможно пополнить новыми требованиями, которые заказчику кажутся простыми, естественными и подразумевавшимися с самого начала. В итоге приходится заново переделывать почти весь проект.

Технология ECO переводит процесс согласования требований на модельный уровень. Диаграммы UML обычно хорошо воспринимаются и заказчиком, и исполнителем. Поэтому, хотя использование визуального языка моделирования в MDA не обязательно, почти 100 % проектов MDA создаются с применением языка UML.

Построение готового приложения происходит в несколько этапов. Сначала строятся модели PIM, затем выполняется их перенос в модели PSM. Доступные па сегодняшний день продукты автоматизируют эти шаги на 50–70 %. Перенос моделей PSM в код конкретного языка программирования автоматизирован почти на 100 %.

Еще один подход, настоятельно рекомендуемый группой OMG при использовании подхода MDA, заключается в выделении логики приложения в отдельную, по возможности платформно-независимую структуру. Достигается это использованием языка объектных ограничений OCL, который сегодня считается частью языка UML. Команды OCL встраиваются непосредственно в модель UML, описывая и уточняя аспекты ее работы. Удобство OCL еще и в том, что его выражения напоминают естественный язык (предложения, записанные на английском). Это позволяет сохранять при построении моделей контакт с людьми (например, представителями заказчика), не являющимися специалистами в области программирования.

При реализации сложной логики далеко не всегда удается обойтись стандартными возможностями языков OCL и UML. Всевозможные манипуляции с наборами объектов, их фильтрация и сортировка решают до 70–90 % потребностей современных проектов. Такие задачи характерны для многих задач автоматизации деятельности организаций. Технология ECO позволяет быстро создавать работающие прототипы, но 10–30 % требований все равно приходится программировать вручную. Для этого используется исходный код на языке Delphi, в который

отображается модель UML, сформированная в среде Delphi. Можно выполнить и обратное преобразование: на основе измененного исходного текста получить готовую модель UML.

Физические объекты, структура которых описана с помощью языка UML, а особенности существования – с помощью языка OCL, функционируют в процессе работы приложения в специально выделенной области оперативной памяти, так называемом объектном пространстве. Технология ECO способна автоматически отображать модель не только в программный код, но и в код на языке SQL. Он генерируется для построения схемы базы данных, хранящей копию объектного пространства. Поддержка разных СУБД является в ECO одной из ключевых технологий. С ее помощью удается сохранять текущее состояние объектного пространства приложения в базе данных и затем, после перезапуска приложения, загружать его и продолжать работу с прерванного места.

Среда Delphi на основе подготовленной модели автоматически формирует схему базы данных с учетом версии и специфики конкретной СУБД.

Она автоматически создает в ней все таблицы, нужные для хранения объектного пространства и взаимосвязей между классами, даже если это требует введения дополнительных таблиц. По мере совершенствования модели следует периодически синхронизировать схему базы данных с внесенными в модель модификациями – для этого достаточно нажать одну кнопку. Такой процесс называется в технологии ECO эволюционным развитием базы данных.

Стыковка объектного пространства с пользовательским интерфейсом реализуется в ECO с помощью дескрипторов. Дескриптор ECO – это стыковочная точка (компонент промежуточного уровня), связывающая с помощью выражений OCL элементы модели UML (классы) с программным кодом и внутренней структурой обычного приложения Delphi. Дескрипторы задают способы формирования наборов объектов в объектном пространстве по критериям, заданным выражениями OCL. Они также обеспечивают доступ к ним элементов пользовательского интерфейса, а программному коду предлагают ссылки на объекты ECO.

5.2 Язык объектных ограничений OCL

Один из самых серьезных и справедливо критикуемых недостатков языка моделирования UML – предоставление разработчику только визуальных средств моделирования. Эти средства абстрактны и поэтому далеко не всегда способны точно и формально отразить тот или иной нюанс функционирования проектируемой системы. В результате проектировщик не находит достаточно выразительных средств, позволяющих уточнить ограничения на применение создаваемых структур, специфицировать способы и нюансы их внутреннего функционирования, ввести условия использования и так далее. Конечно, можно представить эти особенности программным кодом конкретной среды программирования. Однако язык UML нацелен, прежде всего, на создание платформно-независимых моделей. Поэтому применяемый для этого текстовый язык, связанный с моделями UML, как минимум, тоже должен быть независим от операционной системы и среды разработки.

В начале 1990-х гг. в корпорации IBM разрабатывалась методика объектно-ориентированного анализа и проектирования Syntropy. Она включала математический язык текстовых описаний элементов визуальных моделей, который оказался сложен для массового практического использования.

Впоследствии на его основе был создан язык объектных ограничений OCL (Object Constraint Language), который применялся тогда как язык моделирования. Сильной стороной языка OCL оказалась независимость от плат- формы реализации и легкая адаптация к разным средам программирования. Он активно применялся для описания особенностей объектных информаци- онных систем и в 1997 г. вошел в стандарт языка UML 1.1.

До появления языка OCL в системах визуального моделирования при уточнении ограничений на диаграммах UML применялись комментарии или рекомендации на естественном языке. Они нередко трактовались неоднозначно.

Язык OCL снял множество проблем при проектировании моделей UML. Он предоставил разработчику набор формальных средств взаимодействия с объектами создаваемого приложения. Со временем язык OCL перерос границы стандарта UML. Он нередко задействуется как универсальный и, главное, платформно-независимый язык описания бизнес-логики. Например, сценарии OCL можно выделять и настраивать в проекте независимо от

программного кода на конкретном языке.

Язык OCL не является языком программирования. Он предназначен для формального определения выражений, обрабатывающих объекты. Выражение OCL обычно привязано к определенному классу UML и задает множество экземпляров этого класса. Команды OCL выполняют также фильтрацию этого множества или, например, определяют число его элементов.

Язык OCL содержит развитые средства манипуляции над множествами объектов, поэтому он хорошо подходит для решения задач, где обычно применяется язык запросов к реляционным базам данных SQL. Язык OCL позволяет организовывать запросы с большей эффективностью и на более высоком модельном уровне абстракции, нежели язык SQL.

Выражение OCL фактически задает условие, которому должны удовлетворять все экземпляры соответствующего объекта UML. Результатом выполнения выражения OCL является множество объектов, удовлетворяющих этому условию. Условие, которое задается выражением OCL, называется инвариант. Оно представляет собой набор правил или шаблон, накладываемые на описание объекта. Значением инварианта является логическая величина (True или False). Выражения OCL встраиваются в модели UML или задаются в сопроводительной документации.

Язык OCL надежен и безопасен. Стандартом гарантируется, что любое его выражение будет вычислено без побочных эффектов и не окажет влияния на части модели (ни на какие ее состояния, значения или связи). Среда, вычисляющая выражение OCL, просто определяет результирующее значение, которое может впоследствии использоваться (хотя это и не обязательно). Сам язык OCL может применяться для реализации бизнес-логики, управления процессами и других задач только в качестве средства расчета выражений.

Привязка выражения OCL к объекту UML происходит через так называемое объявление контекста. В начале выражения задействуется наименование нужного класса объекта. Может также указываться ключевое слово self (ссылка на текущий объект). Оно допускается, если контекст вычисляемого выражения однозначен, очевиден и может быть связан с выражением автоматически.

5.2.1 Типы данных и операции OCL

В языке OCL используется четыре основных типа данных – значения могут быть целыми, вещественными (с плавающей запятой), логическими и строковыми. Над числами определены стандартные арифметические операции: сложение (знак +), вычитание (–), умножение (*), деление (/). Над значениями всех типов допускаются операции сравнения: меньше (<), больше (>), меньше или равно (<=), больше или равно (>=), не равно (<>), равно (=). Значения логических типов можно обрабатывать с помощью логических операций or (ИЛИ), and (И), not (НЕ), xor (исключающее ИЛИ), а также операции implies. Операция implies представляет собой особую форму логической операции И, результат которой зависит от порядка операндов: X implies Y – это не всегда то же самое, что Y implies X.

5.2.2 Инфиксная форма записи выражений OCL

В язык OCL входит ряд операций, которые являются своеобразными аналогами стандартных функций языка Delphi. Только записываются они не как функции Delphi (имя идентификатора и параметры в круглых скобках), а в так называемой инфиксной записи, которая напоминает форму записи обычных арифметических выражений.

Инфиксная запись выражения подразумевает размещение операндов по обе стороны знака операции и вычисление выражения слева направо без учета приоритетов операций.

Типичное выражение OCL представляет собой цепочку элементов, ссылок и вызовов операций, которые последовательно начиная с левого края выражения и до его правого края обрабатывают значение, полученное к моменту их вызова. Функциональные элементы и операции языка OCL применяются к аргументу, расположенному левее. Они записываются справа от него через точку. Вторым аргументом операции или вторым параметром функции OCL считается элемент, расположенный правее, его заключают в круглые скобки.

Так, для действительных чисел можно использовать операцию abs для определения модуля числа или операцию round для округления. Однако эта операция записывается не слева, как принято в языках программирования, а справа от обрабатываемого значения и через точку. Само

значение, соответственно, указывается слева от имени функции и считается ее первым аргументом. Так как второй параметр для данной операции не нужен, результат ее работы можно передать далее, поставив еще одну точку, вслед за которой можно записать очередную функцию.

Пусть, например, требуется округлить число 123,45 с помощью функции round. Соответствующее выражение OCL записывается следующим образом:

123.45.round

Получить модуль отрицательного числа –123 с помощью функции abs можно так:

–123.abs

Если операция OCL использует два аргумента, второй из них заключается в круглые скобки, как обычный параметр. Например, операция max по выбору максимального из значений x и y запишется следующим образом:

x.max(y)

Для логической обработки язык OCL предлагает несложную форму условного оператора. Такой оператор фактически представляет собой вычисляемое выражение и записывается следующим образом:

if условие then выражение 1 else выражение 2 endif

Если условие истинно, то вычисляется выражение l, и результатом условного оператора считается его значение. В противном случае вычисляется выражение 2 и результатом считается уже его значение.

5.2.3 Последовательности доступа к объектам в языке OCL

Выражение OCL возвращает результат, который может быть: экземпляром одного из базовых типов; экземпляром одного из классов модели; метаинформацией (сведениями об определенном типе данных); набором или коллекцией экземпляров одного класса. Элементы такой коллекции могут быть упорядочены или неупорядочены. Внутри коллекции допускаются одинаковые экземпляры. Если к обычным, скалярным, значениям операции OCL применяются через точку (символ .), то к коллекциям – через комбинацию символов ->.

Для доступа к полям определенного объекта в языке OCL применяется запись, в которой сначала следует имя объекта, затем точка и имя поля объекта. Например, если имеется объект Computer (экземпляр класса Computer), а в этом классе описано поле Mouse, то обращение к полю Mouse записывается следующим образом:

Computer.Mouse

Если у поля Mouse (экземпляра класса Mouse), в свою очередь, имеются внутренние поля, то и к ним можно обращаться, продолжая запись через точку:

Computer.Mouse.Position

Поля объектов в выражении OCL могут быть не скалярными значениями, а коллекциями. Для того чтобы получить (выбрать) все существующие в программе на данный момент экземпляры определенного типа, используется стандартная операция allInstances. Например, все созданные в приложении экземпляры класса Computer можно получить следующим выражением:

Computer.allInstances

Результатом такого выражения будет не один объект, а коллекция, набор экземпляров класса Computer. Соответственно, запись

Computer.allInstances.Mouse

формирует коллекцию значений – экземпляров класса Mouse, входящих, в свою очередь, в состав класса Computer (физически – в текущее множество экземпляров этого класса).

5.2.4 Операции над коллекциями

Рассмотрим основные операции языка OCL, которые можно применять к коллекциям. Такие операции иногда называют итераторами. Они чаще всего задействуются в моделях ECO для описания логики работы программ и формирования наборов записей, отображаемых на экране. Эти операции записываются с использованием операции ->.

5.2.4.1 Стандартные операции

Для выполнения операций над коллекциями язык OCL предлагает ряд стандартных

функций:

- функция Size возвращает число элементов коллекции;
- функции MinValue, MaxValue возвращают, соответственно, минимальное или максимальное значение из набора;
- функция Sum подсчитывает сумму всех элементов набора;
- функция Count определяет число элементов коллекции, удовлетворяющих условию, заданному в качестве параметра. Например, выражение Count (Name='Сергей') вычислит число элементов коллекции, у которых в поле Name содержится строка 'Сергей';
- функции isEmpty, NotEmpty возвращают значение True, если, соответственно, в анализируемом наборе нет ни одного элемента или есть хотя бы один элемент.

Функция, применяемая к коллекции, может возвращать скалярный результат (единственное значение) или вектор (набор значений). Результаты работы функций MinValue, MaxValue, Count и некоторых других относятся к последнему типу. Результатом может быть коллекция из нескольких одинаковых значений, каждое из которых соответствует объекту исходной коллекции, отвечающему заданному условию. Если, например, в исходном наборе несколько минимальных величин с одинаковыми значениями, все они попадут в результирующую коллекцию. Чтобы гарантированно получить скалярный результат, надо применить к итоговому набору операцию First.

5.2.4.2 Операция Select

Операция Select позволяет выбрать в коллекции подмножество объектов, свойства которых отвечают заданному условию. Это условие указывается после ключевого слова Select в круглых скобках.

Пусть, например, в классе Computer (компьютер) имеется свойство Memory (объем оперативной памяти). Отбор всех объектов класса Computer, у которых объем оперативной памяти больше или равен 512 мегабайт (512 единицам, принятым в модели по умолчанию), выполняет следующее выражение OCL:

Computer.allInstances->Select(Memory >= 512)

В дальнейшем по отношению к результирующей коллекции (в инфиксном порядке) можно применять другие допустимые операции OCL, в том числе и саму операцию Select. Например, возможно такое обращение к полям текущего подмножества объектов:

Computer.allInstances->Select(Memory >= 512).Mouse

Это выражение отбирает объекты, представляющие компьютерные мыши (свойство Mouse), которыми оснащены компьютеры с оперативной памятью не менее 512 мегабайт.

5.2.4.3 Операция Reject

Операция Reject отбирает в выходную коллекцию только те объекты, которые не удовлетворяют заданному условию. По смыслу она противоположна операции Select. В примере

Computer.allInstances->Reject(Memory < 512)

будет получен тот же результат, что и при использовании выражения:

Computer.allInstances->Select(Memory >= 512).

5.2.4.4 Выделение элементов коллекции

Для получения первого элемента в коллекции применяется операция First. Для получения последнего элемента в коллекции применяется операция Last. Выделение объекта по его номеру в коллекции выполняет операция At – номер нужного объекта задается в качестве параметра (нумерация начинается с единицы).

Например, следующее выражение получает последний элемент из коллекции компьютеров:

Computer.allInstances->last

Получить название (свойство Name) процессора (свойство Processor) первого элемента текущей коллекции компьютеров можно следующим образом: Computer.allInstances->first.Processor.Name

Получить объект, представляющий мышь пятого компьютера текущего набора, можно следующим образом:

Computer.allInstances->At(5).Mouse

5.2.4.5 Упорядочение набора

Применение операций выделения первого и последнего элементов не всегда имеет смысл, если исходная коллекция не упорядочена. Упорядочение наборов данных производится операциями OrderBy и OrderDescending. Они выполняют сортировку по заданному параметру, причем операция OrderDescending выполняет сортировку «в порядке убывания» – способом, противоположным OrderBy. Конкретный способ сортировки коллекций с помощью операции OrderBy определяется реализацией.

Например, отсортируем набор объектов-компьютеров по именам: Computer.allInstances->OrderBy(Name)

5.2.4.6 Логические итераторы

Проверка выполнения логического условия для всех элементов коллекции осуществляется с помощью итератора ForAll. Пусть, например, в классе Computer имеется свойство Mouse (отдельный класс), а у него, в свою очередь, имеется логическое свойство isMiddleButton (наличие средней кнопки мыши). Тогда выражение:

Computer.allInstances->ForAll(Mouse.IsMiddleButton) вернет значение True, если все без исключения компьютеры коллекции оснащены трехкнопочными мышами.

Итератор Exists, схожий по смыслу с итератором ForAll, возвращает значение True, если хотя бы один из элементов коллекции соответствует заданному условию:

Computer.allInstances->Exists(Mouse.IsMiddleButton)

5.2.4.7 Операции для работы со строками

Для работы со строками в языке OCL применяются следующие операции. Соединение двух строк в одну (символ +) может также задаваться ключевым словом concat. Строка-параметр присоединяется к исходному операнду справа. Например, следующее выражение к имени первого компьютера в коллекции прибавляет строку ': ноутбук':

Computer.allInstances->first.Name.Concat(': ноутбук')

Результатом выделения подстроки (ключевое слово Substring) является строка, выделенная из исходного операнда. Дополнительные параметры задают номера первого и последнего символов, включаемых в подстроку (нумерация начинается с единицы). Например, выражение

'компьютер'.Substring(2,4)

Вернет подстроку «омп».

Функции ToLower, ToUpper преобразуют все символы строки операнда, соответственно, к нижнему или верхнему регистру.

Функция strToint преобразует исходную строку в целочисленное значение.

5.2.4.8 Работа с датами

Даты представляются в языке OCL в формате #гггг-мм-дд, где гггг – четырехзначная запись года, мм – номер месяца, а дд – число в месяце. Схожим образом записываются и временные константы – в формате #чч:мм:сс. Здесь чч означает часы, мм – минуты, сс – секунды. Константы, записанные в таком виде, можно использовать в операциях сравнения. Например:

Computer.allInstances->Select(MyDate > #2006.01.01)

Users.allInstances->Select(ConnectTime = #23:59:00)

Чтобы проверить, лежит ли дата в указанном диапазоне, следует воспользоваться функцией inDateRange. Двумя ее параметрами выступают нижняя и верхняя границы дат.

5.3 Возможности технологии ECO
5.3.1 Введение в технологию ECO

Технология ECO – это реализация концепции архитектуры, управляемой моделью. Она позволяет полностью создать приложение с помощью языка UML.

Технология ECO включает в себя набор объектов моделирования .NET и работает только на платформе .NET. В ней используются диаграммы UML как на фазах начального проектирования приложения, так и на этапе эксплуатации программы. Это достигается за счет встраивания модели в приложение.

Почти весь процесс создания готового приложения в рамках проекта ECO осуществляется манипулированием визуальной моделью, представленной в виде диаграмм UML. При этом удается избежать ручной модификации исходного кода и минимизировать обращения к проектировщику пользовательского интерфейса – весь исходный код приложения автоматически генерируется на основе визуальной модели. Ручная модификация кода нужна обычно для тонкой настройки приложения на дополнительные требования пользователя.

Центральной концепцией программы ECO считается объектное пространство. Всевозможные аспекты создания и поведения объектов в этом пространстве задаются с помощью платформно-независимого языка объектных ограничений OCL.

Объектное (или модельное) пространство ECO – это область памяти, в которой существуют и взаимодействуют объекты ECO текущего проекта.

В системе Delphi 2006 поддерживается третья версия технологии ECO III. Она состыкована со средой моделирования Borland Together, выпускаемой как самостоятельный продукт. Технология ECO III позволяет проектировать не только статическую структуру приложения, иерархию классов, но и его поведение, программную логику. Для этого задействуются так называемые машины состояний (описывающие их диаграммы состояний появились в версии языка UML 2.0). В итоге почти вся разработка сложного приложения выполняется в дизайнере диаграмм UML, встроенном в систему Delphi. На базе созданных таким образом моделей автоматически генерируется полнофункциональный исходный код приложения. В случае ручной модификации исходных текстов структура модели также автоматически подстраивается под внесенные изменения. Такая двунаправленная технология синхронизации модели и кода получила в среде Delphi название LiveSource.

Технология ECO поддерживает не только этап построения приложения и модели ECO, но и этап выполнения программы. При запуске приложения ECO создается объектное пространство ECO, которое называется ECO Space. В этом пространстве хранится вся необходимая метаинформация о модели. В рамках объектного пространства происходит создание и уничтожение экземпляров элементов ECO. Содержимое пространства ECO можно автоматически связать с данными в файлах или базах данных.

Технология ECO хорошо подходит для построения масштабных корпоративных приложений. Она значительно сокращает время разработки приложений, в которых активно задействуются базы данных с множеством таблиц и сложными взаимосвязями и большое число экранных форм, создание которых вручную является трудоемкой рутинной задачей. В технологию ECO встроены средства поддержки реляционной модели баз данных. Они автоматически преобразуют модель UML в описание схемы базы данных, генерируют сценарии SQL, которые создают и модифицируют такие схемы, и выполняют другие необходимые операции.

5.3.2 Модель ECO

Модель может объединять несколько взглядов на объект, каждый из которых уточняет ту или иную особенность его существования. Модель компьютерной программы (в частности, модель ECO) отличается от физических и математических моделей тем, что на ее основе должен быть автоматически сгенерирован безошибочный и работоспособный исходный текст на заданном языке программирования. Это предъявляет повышенные требования к полноте модели ECO, ее точности и непротиворечивости.

Процесс подготовки модели удается упростить с помощью набора готовых объектов ECO. Они прозрачно обеспечивают взаимосвязь между программным кодом и элементами модели. В

результате проектирование и развитие архитектуры всего приложения существенно упрощается.

5.3.3 Пространство имен ECO

Познакомимся с организацией пространства имен ECO. В рамках иерархии классов Delphi все классы ECO входят в единое пространство имен Borland.Eco. Оно, в свою очередь, делится на шесть больших категорий.

Категория Borland.Eco.ObjectRepresentation содержит средства стандартного представления объекта ECO во внешних программах. Каждый из прикладных объектов ECO состоит из элементов (внутренние поля, методы). К каждому из этих элементов можно обращаться через стандартный интерфейс Element. Любой элемент объекта, доступный через этот интерфейс, может быть представлен в виде стандартного объекта .NET. Для этого задействуется его свойство AsObject.

Категория Borland.Eco.Handles содержит компоненты поддержки модельного пространства на этапе проектирования. Они связывают объекты ECO, доступные через интерфейсы пространства имен ObjectRepresentation, с элементами пользовательского интерфейса.

Категория Borland.Eco.UmIRt содержит средства доступа к модели UML в процессе работы программы. Компоненты ECO базируются на классах .NET, поэтому к ним можно обращаться универсальным способом, принятым в .NET. В частности, для этого задействуется метод AslObject, предоставляющий для доступа к объекту интерфейс IObject платформы .NET.

Категория Borland.Eco.Subscription содержит средства уведомления об изменении значений объектов ECO по технологии «издатель – подписчики».

Категория Borland.Eco.Persistence содержит средства автоматического сохранения содержимого объектного пространства ECO в файлах и базах данных.

Категория Borland.Eco.Services – это среда поддержки внутренних механизмов технологии ECO. Она занимается вычислением значений выражений OCL, вносит модификации в модель, контролирует ее целостность.

5.4 Разработка приложений на основе ECO

5.4.1 Этапы создания приложения по технологии ECO

Технология создания приложения ECO состоит из следующих этапов.

1. Формируется модель UML будущего приложения. Создаются классы, описываются их атрибуты и методы, настраиваются взаимосвязи.
2. В проект добавляются и настраиваются невизуальные компоненты, связывающие созданную модель UML с прикладной частью проекта.
3. Проектируется пользовательский интерфейс. Задействуются компоненты, обеспечивающие связь интерфейса с моделью UML.
4. Создается переносимая логика приложения на языке OCL. Элементы управления связываются с выражениями OCL для выполнения типичных стандартных действий (добавление, редактирование и удаление объектов: ECO).
5. Пространство ECO связывается с базой данных. В ней будет долговременно храниться его копия: все объекты ECO и связи между ними. Эта последовательность в ходе расширения функциональных возможностей приложения многократно повторяется. Но все вносимые в проект изменения, что принципиально важно, выполняются, начиная с модели, а не с исходного кода или пользовательского интерфейса.

5.4.2 Создание простого MDA-приложения

Рассмотрим пример создания простого MDA-приложения. Имеются две сущности: *Кафедра* и *Преподаватель*. Представим экземпляры этих сущностей в таблицах формы. В этих же таблицах будет возможность модификации значений отдельных полей представленных в них объектов (экземпляров *Кафедра* и *Преподаватель*).

Создается пустая заготовка приложения ECO командой File>New>Other. Выберем значок ECO WinForms Application во вкладке Delphi for .NET Projects (рисунок 5.1).

В диалоговом окне введем имя проекта (например, projDeanOffice) и его

местоположение (каталог). Нажмем кнопку *OK*.

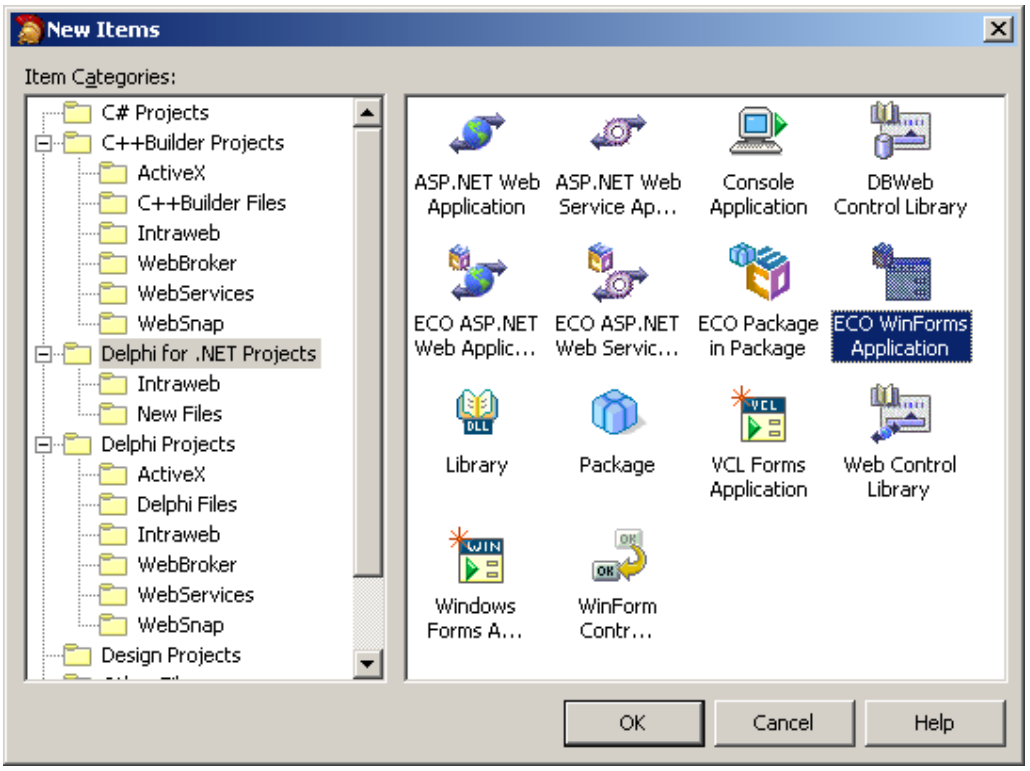


Рисунок 5.1 - Создание заготовки проекта ECO

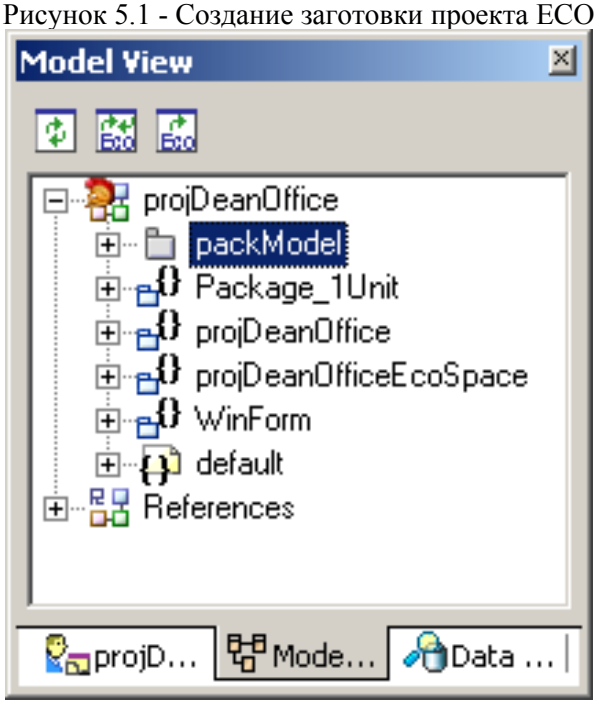


Рисунок 5.2 - Окно просмотра модели

Среда Delphi сформирует пустую заготовку приложения ECO и откроет окно *Проектировщика*. В нем расположена пустая начальная форма приложения. Структура автоматически созданной модели (пустой заготовки) доступна в окне просмотра модели, открываемом командой View>Model View либо выбором вкладки Model View в правом верхнем окне (рисунок 5.2). В этом окне в дополнение к самому проекту (projDeanOffice) и главной форме (Win-Form) можно увидеть еще несколько автоматически добавленных элементов. Среди них имеются следующие:

- элемент projDeanOfficeEcoSpace представляет объектное пространство ECO. Это основное хранилище, в котором располагаются экземпляры классов создаваемой модели во время работы программы. Это простран- ство также ответственно за хранение объектов ECO, например в базе дан- ных или файле XML;

- элемент Package_1 представляет пакет классов UML, которые мы будем создавать.

Переименуем элемент Package_1 в удобное для нас название packModel.

5.4.2.1 Создание модели UML

Для создания модели дважды щелкнем мышью на строке packModel в окне просмотра модели. Откроется окно packModel [diagram] диаграммы классов ECO. Это окно напоминает окно построения обычных диаграмм классов UML. Только при работе с ним применяются элементы, специфичные именно для технологии ECO.

Разместим на диаграмме первый класс, отражающий сущность *Деканат*. Для этого выберем на палитре инструментов Tool Palette инструмент ECO Class в категории UML ECO Class Diagram и щелкнем в подходящей точке пространства моделирования. В ней появится графический элемент, изображающий класс.

Назовем класс clChair (*Кафедра*). Пробелы в имени класса ECO не допускаются.

Добавим атрибуты класса командой контекстного меню Add>Attribute. Создадим таким образом три атрибута: *Название кафедры*, *Ф. И. О. заведующего кафедрой*, *Ф. И. О. секретаря кафедры* – ChairName, ChairHeadSNP и ChairSecrSNP соответственно (рисунок 5.3). Для задания типа атрибута выделим нужный атрибут и в окне Properties установим необходимое значение поля Type в категории General. В данном случае все три атрибута имеют тип String.

Переименуем названия класса и атрибутов в понятные названия на русском языке. Для этого в свойстве Alias (категория General) класса и его атрибутов введем русскоязычные названия.

По аналогии добавим к модели еще один класс clLecturer (*Преподаватель*) с атрибутами LecturerSNP (*Ф. И. О. преподавателя*) и LectAcadDegree (*Ученая степень*) типа String. Переименуем элементы нового класса на диаграмме, изменив значения свойства Alias.

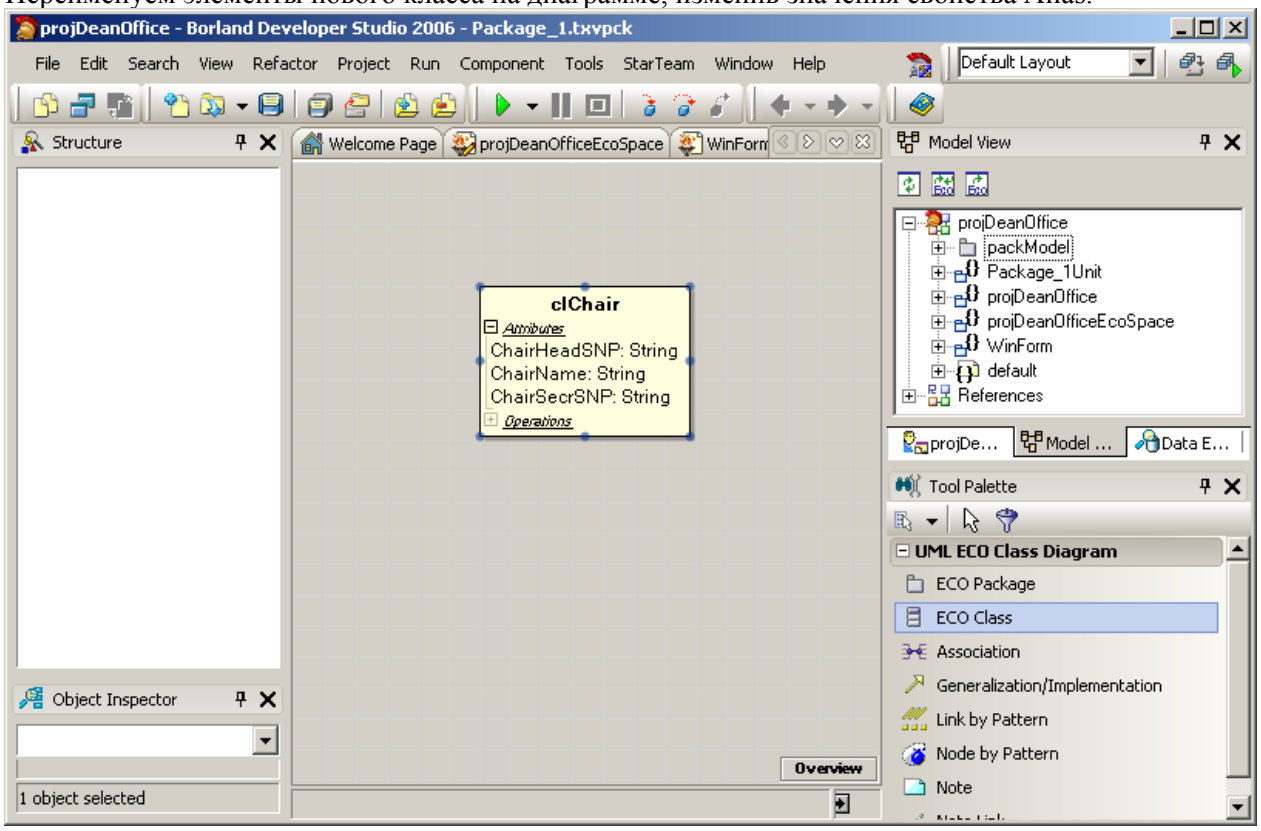


Рисунок 5.3 - Добавление класса clChair с его атрибутами

Настроим связи между созданными классами. Эта связь будет представлять отношение ассоциации. Выберем на палитре инструментов инструмент Generalization/Implementation. Щелкнем мышью на представлении класса *Кафедра*, протянем связь к классу *Преподаватель* и снова щелкнем мышью. В результате между двумя классами сформируется ассоциативное отношение.

Настроим мощность ассоциативного отношения. В нашем случае одному экземпляру

класса *Кафедра* соответствует множество экземпляров класса *Преподаватель* (от 1 и более). Для этого в окне Properties выделенной связи выберем категорию End1, соответствующую классу *Кафедра*, и в поле Multiplicity установим значение 1, а в категории End2 (для класса *Преподаватель*) этому же полю – значение 1..*. Теперь дадим сторонам связи назва- ния. В свойство Name для сторон End1 и End2 введем roleChair и roleLecturers соответственно (рисунок 5.4). То есть мы назвали роли каждого класса в ассоциативной связи.

На этом этапе создание простейшей модели закончено. Теперь надо организовать связь модели с пользовательским интерфейсом.

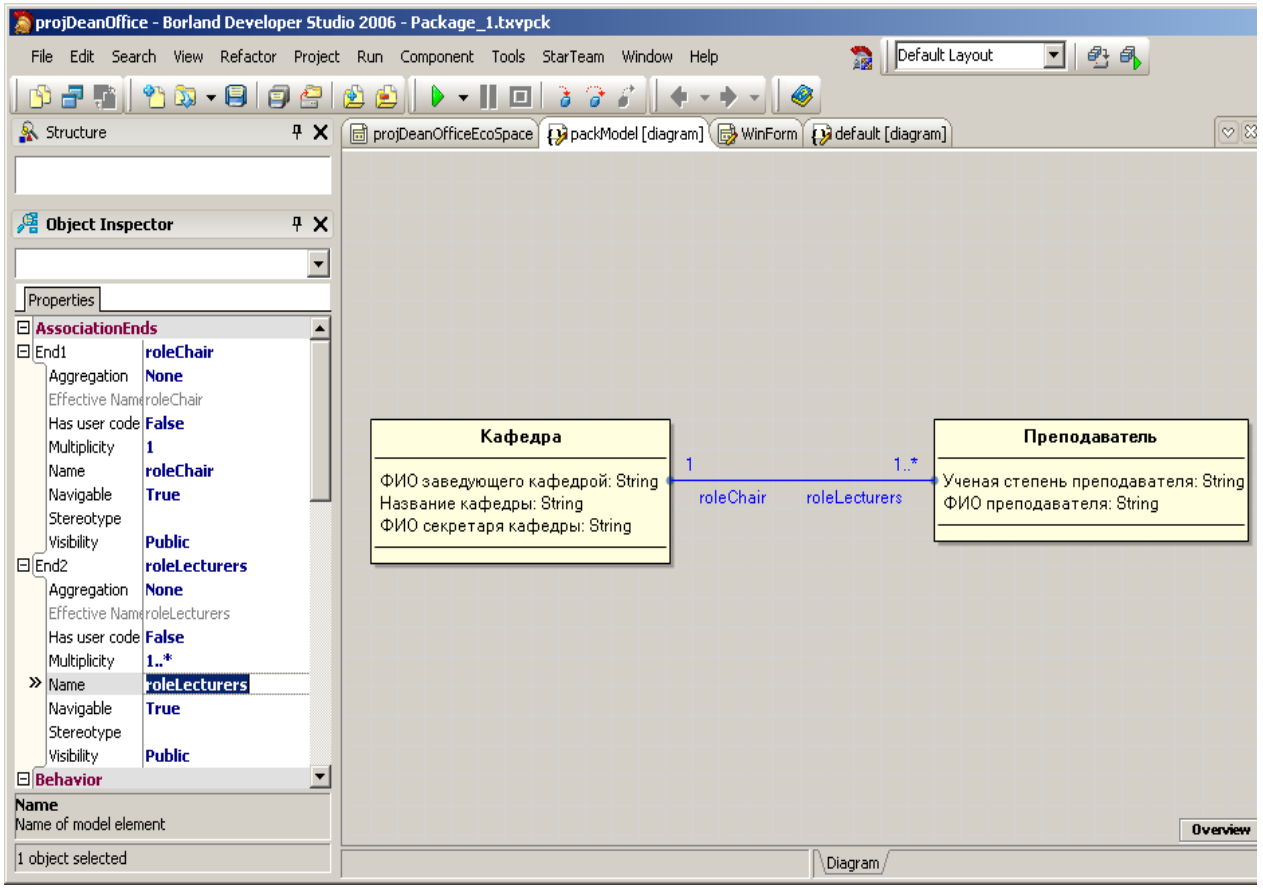


Рисунок 5.4 - Настройка связи между классами

5.4.2.2 Создание интерфейса

Перейдем к окну *Проектировщика* для главной формы проекта Win-Form (вкладка Design). Переименуем стандартное название главной формы.

Назовем ее wfMainForm, а сам класс TWinForm переименуем в TMainForm в свойстве Name категории Design. Свойство Text (категория Appearance) отвечает за название заголовка окна, введем в него строку, например *Главная форма*.

Для представления таблицы с объектами ECO на форме воспользуемся готовым компонентом DataGrid из категории палитры инструментов Data Controls. Поместим этот компонент на форму и дадим ему название dgChair (в свойстве Name). Эта таблица будет отвечать за представление экземпляров класса *Кафедра* (clChair). Исходно таблица должна быть пуста. В свойстве CaptionText (заголовок таблицы) введем название *Кафедры*.

Добавим еще одну таблицу dgLecturer, которая в готовом приложении будет отвечать за отображение экземпляров класса *Преподаватель* (clLecturer). В свойстве CaptionText введем название *Преподаватели*.

Для работы с таблицами в простом приложение потребуется пять операций: добавление и удаление данных в двух таблицах и сохранение копии ECO пространства из оперативной памяти в базу данных. Редактирование данных будет осуществляться непосредственно в полях таблиц. Добавим на форму пять кнопок – экземпляров класса Button (рисунок 5.5).

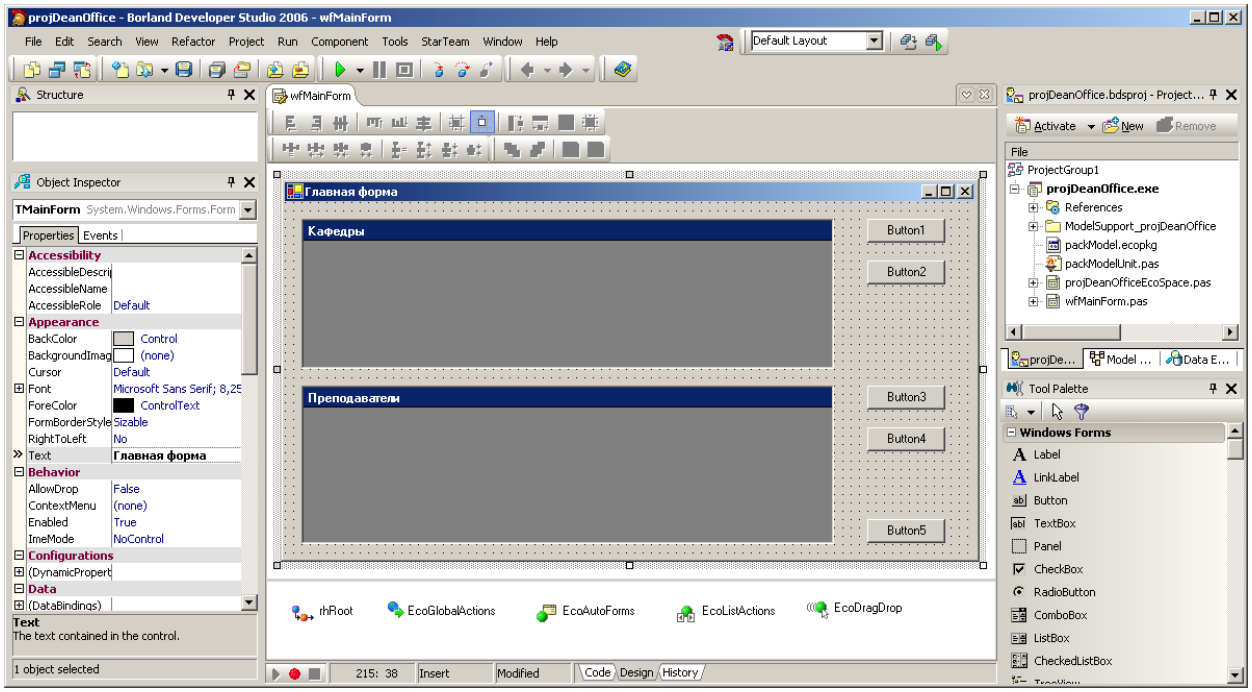


Рисунок 5.5 - Добавление компонентов пользовательского интерфейса

5.4.2.3 Связывание интерфейса с моделью

Связь пользовательского интерфейса с моделями ECO обычно происходит через компонент ExpressionHandle. Он доступен в категории палитры инструментов Enterprise Core Objects. Через подобные идентификаторы (де-скрипторы) организуется доступ к объектам модели и пространства ECO во время работы программы.

Идентификаторы ECO связываются друг с другом в цепочки. Во главе такой цепочки расположен корневой идентификатор. Он задает общий контекст работы (доступ к объектному пространству) для всех остальных идентификаторов ECO в программе.

Корневой идентификатор добавляется к проекту автоматически (по умолчанию он называется rhRoot). Все остальные идентификаторы ECO добавляются и настраиваются вручную, что связывает пространство ECO с элементами пользовательского интерфейса с помощью типовых механизмов связывания .NET.

Добавим к проекту компонент ExpressionHandle. Он отображается в нижней части окна Проектировщика, под формой, где уже имеется набор компонентов ECO. Назовем его ehChair, введем это имя в свойство Name категории Design.

В свойстве RootHandle этого дескриптора выберем значение rhRoot – ссылку на родительский, автоматически созданный корневой идентификатор.

Так задают место данного идентификатора в цепочке доступа к объектному пространству ECO.

В свойстве DataSource таблицы dgChair выберем имя ehChair в списке доступных идентификаторов ECO.

Настроим таблицу *Преподаватель* так, чтобы она отражала список преподавателей, принадлежащих выбранной кафедре в первой таблице. Для начала добавим в проект дескриптор CurrencyManagerHandle (из категории Enterprise Core Objects) и дадим ему имя cmhChair, так как этот компонент будет указывать на текущую строку таблицы *Кафедра*. Свяжем дескриптор cmhChair с таблицей dgChair через свойство BindingContext. Теперь он отслеживает выделенную строку этой таблицы.

Зададим ссылку на объект ehChair в свойстве RootHanler, определяющим корневой идентификатор ECO.

Добавим к проекту еще один компонент ExpressionHandle. Назовем его ehLecturer. Дескриптор ehLecturer будет обращаться к экземплярам класса *Преподаватель*.

Зададим ссылку на объект cmhChair в свойстве RootHanler.

Потребителем объектов, предоставляемых дескриптором ehLecturer, будет вторая

таблица. В ее свойстве DataSource выберем ссылку на объект ehLecturer.

Приступим к настройке элементов пользовательского интерфейса. Организуем с помощью визуальных средств Delphi создание новых экземпляров класса *Кафедра* и добавление их в таблицу. Выделим в окне *Проектировщика* кнопку Button1. В свойстве EcoListAction (Список стандартных действий ECO) выберем значение Add (*Добавить объект*).

Кнопка Button1 автоматически получила название Add. Переименуем кнопку. Для этого воспользуемся компонентом EcoListActions (он добавляется в проект по умолчанию и находится в нижней части окна *Проектировщика*), в его свойстве CaptionAdd введем *Добавить*.

Объект ECO, который мы хотим добавить к проекту (сделать доступным через элементы управления на форме), задается в свойстве RootHandle. В нем надо выбрать подходящий поставщик объектов ECO, в нашем случае – идентификатор ehChair.

Свяжем результат действия кнопки (созданный экземпляр класса *Кафедра*) с визуальным элементом, отображающим этот экземпляр, в нашем случае – с таблицей dgChair. Для этого в свойстве BindingContext (контекст связывания) выберем имя dgChair (рисунок 5.6).

Теперь добавим операцию удаления экземпляров класса *Кафедра*. Реализуем это действие по аналогии с операцией добавления. Выделим в окне *Проектировщика* кнопку Button2. В свойстве EcoListAction выберем значение Delete. Зададим идентификатор ehChair в свойстве RootHandle. Свяжем результат действия кнопки с визуальным элементом – таблицей dgChair. Для того в свойстве BindingContext у кнопки Delete выберем имя dgChair. Перейдем к компоненту EcoListActions и в его свойстве Caption Delete введем *Удалить*.

Настройка кнопки Button3 (будет отвечать за добавление экземпляра класса *Преподаватель*). Значения свойств кнопки: EcoListAction – Add; Roo- tHandle – ehLecturer; BindingContext – dgLecturer. Перейдем к компоненту EcoListActions и в его свойстве Caption Add введем *Добавить*.

Настройка кнопки Button4 (будет отвечать за удаление экземпляра класса *Преподаватель*). Значения свойств кнопки: EcoListAction – Delete; RootHandle – ehLecturer; BindingContext – dgLecturer. Перейдем к компоненту EcoListActions и в его свойстве Caption Delete введем *Удалить*.

Кнопка Button5 будет отвечать за обновление БД. Назовем ее Сохранить. В свойстве EcoAction (категория ECO|GUI) выберем UpdateDatabase.

Таким образом, по нажатию этой кнопки выполняется синхронизация содержимого пространства ECO в оперативной памяти и его копии в базе данных.

После каждого нажатия кнопки *Сохранить* содержимое таблицы сохраняется в БД. После нового запуска программы в таблицу пользовательского интерфейса автоматически загружается содержимое модели, сохраненное при последнем выполнении команды UpdateDatabase.

На данном этапе связывание интерфейса с моделью закончено (рисунок 5.7).

Действия, которые выполняет дескриптор в программе, определяются значением свойства Expression. В это свойство вводится выражение языка OCL, которое определяет схему создания объектов модели ECO.

Используем дескриптор ehChair для обращения к экземплярам класса

Кафедра. Введем в свойство Expression объекта ehChair строку clChair.allInstances. Иначе это можно сделать с помощью редактора OCL выражений. Введенное выражение задает доступ ко всем текущим экземплярам класса *Кафедра* в объектном пространстве. В таблице, показанной на форме в окне Проектировщика, сразу же отобразятся поля класса *Кафедра* (Chair- HeadSNP, ChairSecrSNP и ChairName). Дескриптор ehChair становится поставщиком данных нашей модели ECO для первой таблицы.

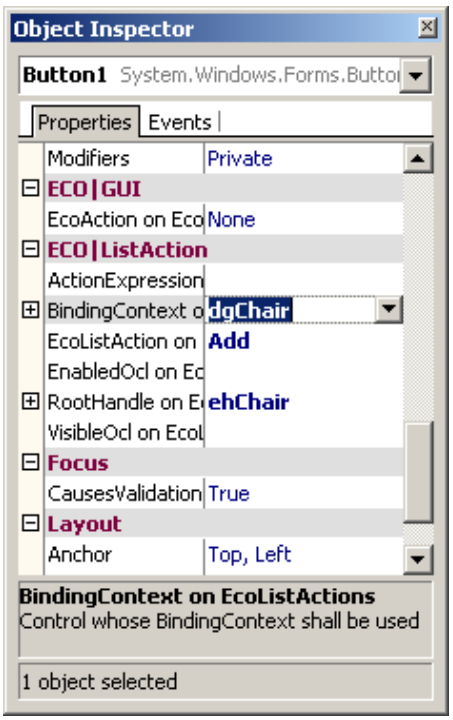


Рисунок 5.6 - Настройка кнопки добавления экземпляра класса Кафедра

5.4.2.4 Создание логики на OCL

В качестве выражения OCL объекта ehLecturer введем строку self.roleLecturers в свойство Expression. Здесь roleLecturers – это введенное нами при построении модели имя роли класса *Преподаватель* (clLecturer) в его ассоциативной связи с классом *Кафедра* (clChair). Это выражение определяет группу преподавателей, принадлежащих кафедре, которая выбрана в первой таблице. Дескриптор ehLecturer становится поставщиком данных для второй таблицы (рисунок 5.8).

Дадим заголовкам полей таблиц русскоязычные названия. Выберем на форме таблицу dgChair. Щелчком на кнопке с многоточием в строке свойства TableStyles откроем окно редактора коллекции стилей таблицы DataGridTableStyle Collection Editor. Создадим новый стиль таблицы, нажав на кнопку Add (рисунок 5.9).

Теперь настроим оставшиеся два элемента коллекции. Эти столбцы будут называться: *Ф. И. О. завкафедрой* и *Ф. И. О. секретаря кафедры*. Нажмем кнопку *ОК* в обоих открытых окнах и убедимся, что поля таблицы получили новые названия.

По аналогии настроим стиль таблицы *Преподаватели*. В ней отобразим два поля: LecturerSNP и LectAcadDegree. Назовем их *Ф. И. О. преподавателя* и Ученая степень.

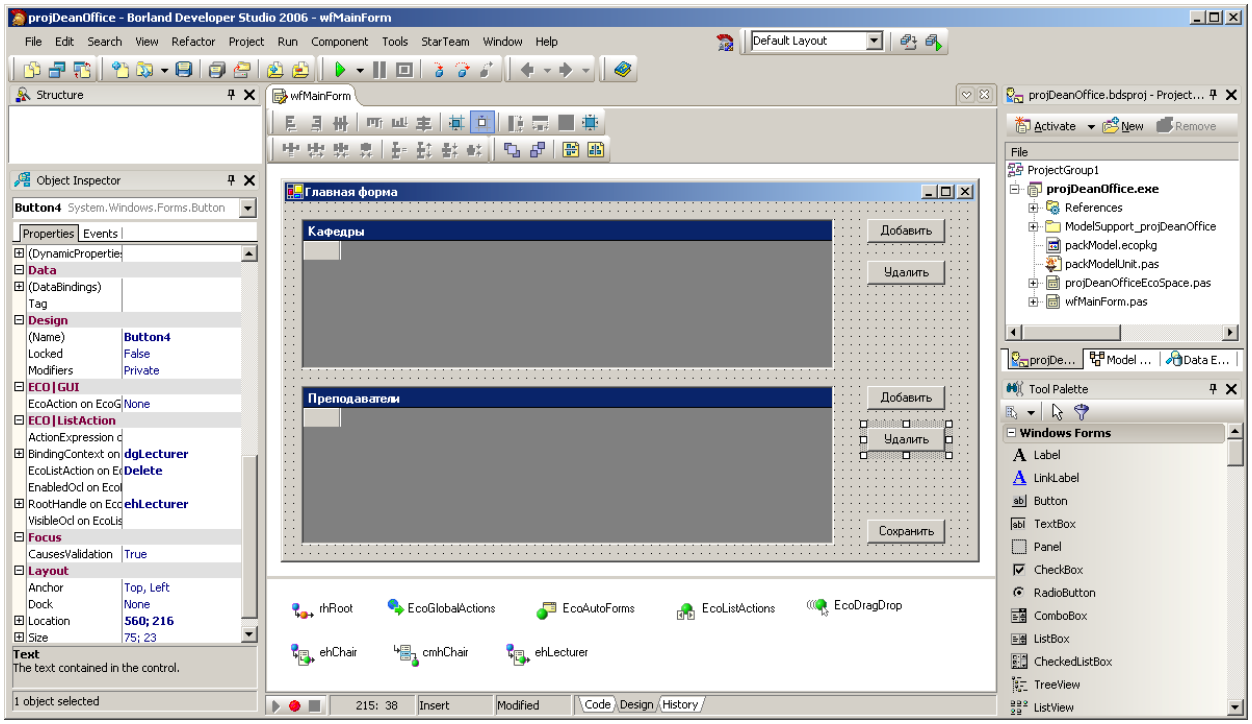


Рисунок 5.7 - Настроенный пользовательский интерфейс приложения

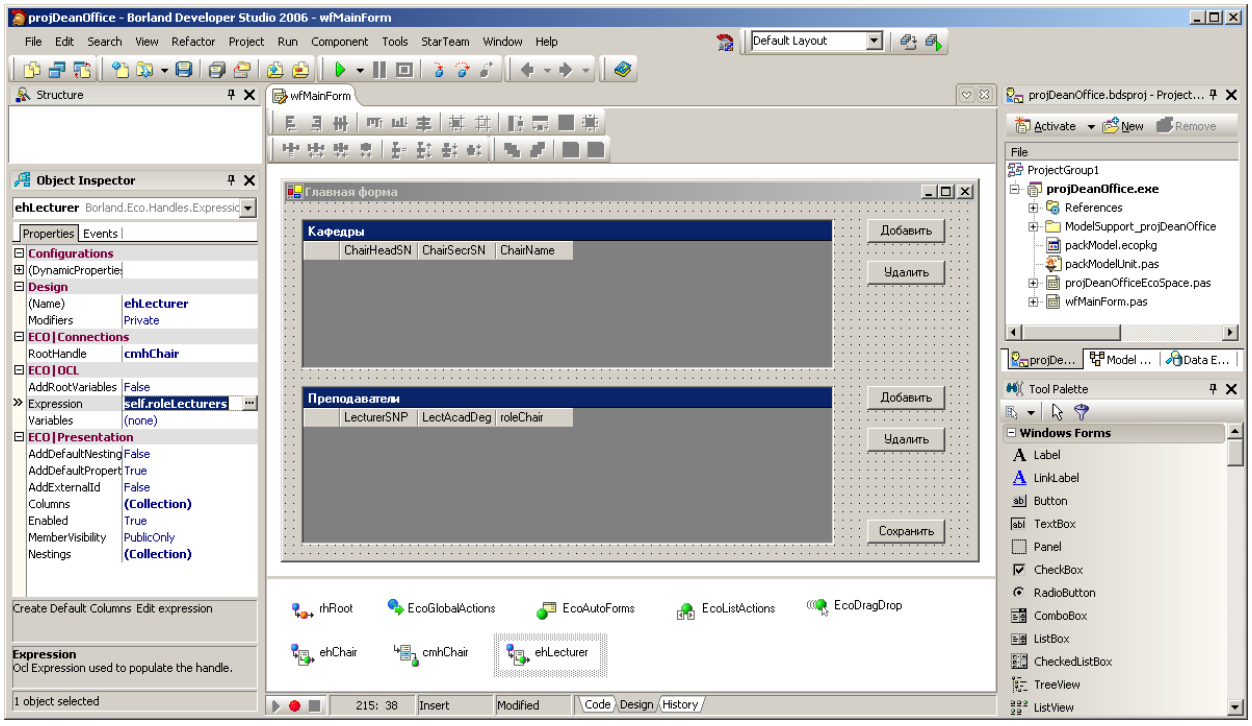


Рисунок 5.8 - Настройка OCL-выражений для взаимосвязанных таблиц

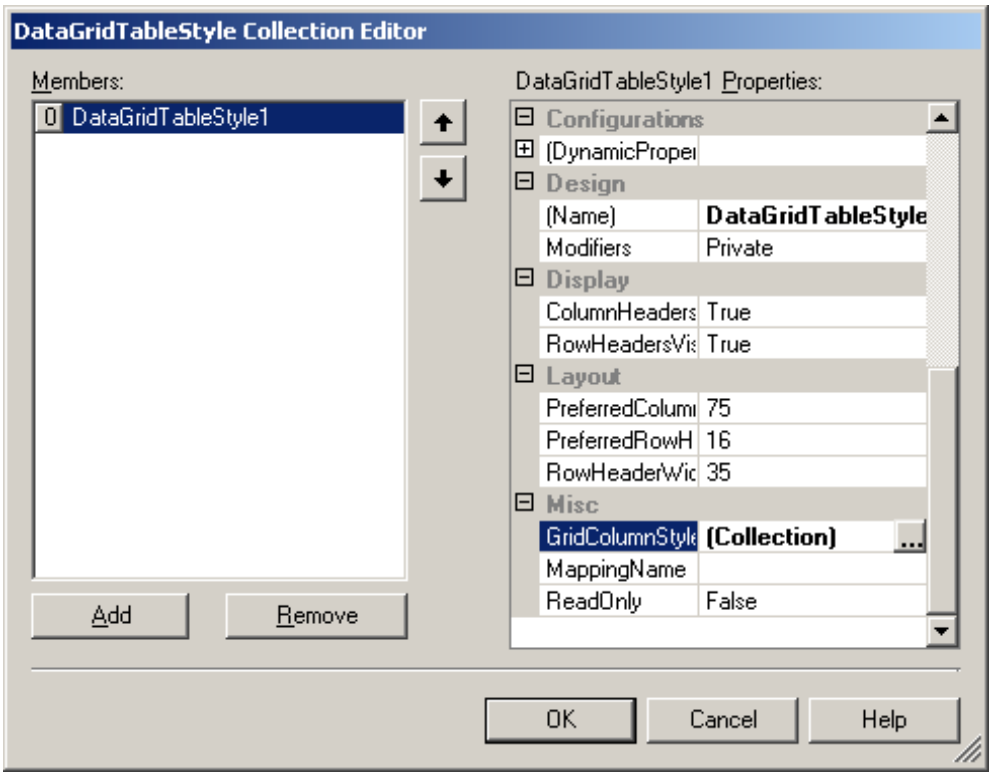


Рисунок 5.9 - Коллекция таблиц компонента gdChair

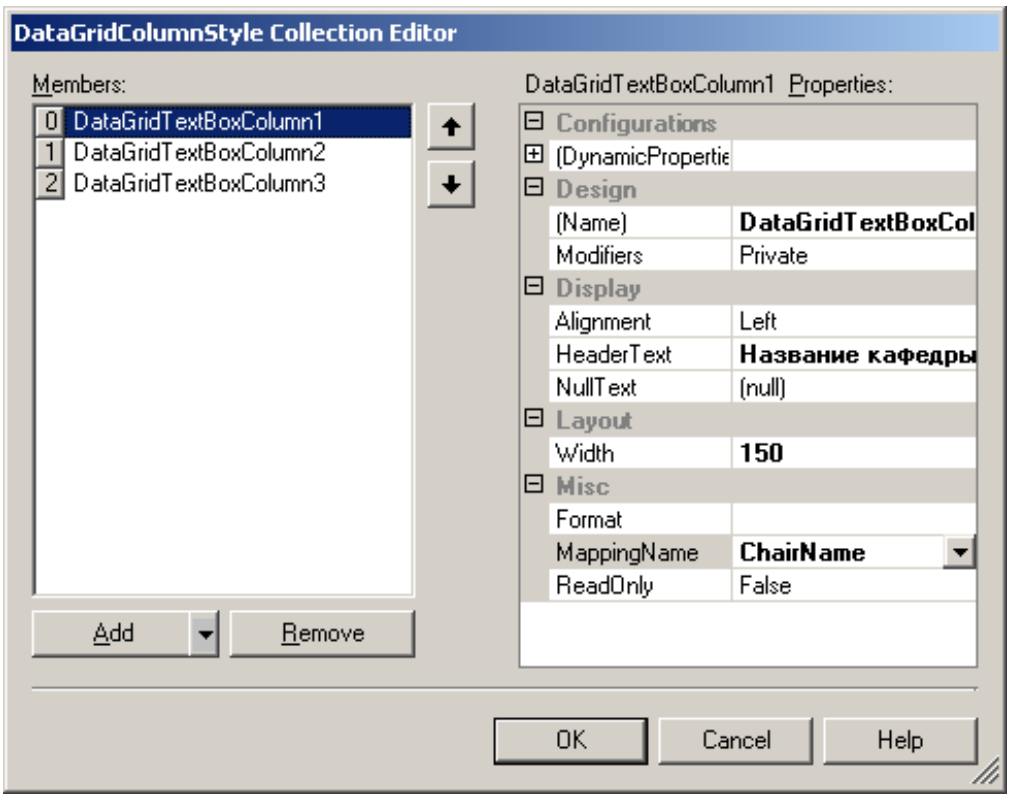


Рисунок 5.10 - Настройка элемента коллекции столбцов

Запустим программу и убедимся, что при нажатии кнопки *Добавить в таблице* автоматически формируются новые строки, отражающие содержимое новых экземпляров класса из объектного пространства ECO. В них можно ввести нужные значения.

План конспекта:

1. Использование архитектуры, управляемой моделью.
2. Язык объектных ограничений OCL.
3. Возможности технологии ECO.
4. Разработка приложений на основе ECO.

Работа с литературой:

Рекомендуемые источники информации (№ источника)			
Основная	Дополнительная	Методическая	Интернет-ресурсы
1-3	1-3	1-3	1-2

5. ВОПРОСЫ К ЭКЗАМЕНУ

Вопросы к экзамену (3 семестр)

Вопросы (задача, задание) для проверки уровня обученности

Знать

1. Основные этапы развития технологии разработки. Этап 1 - «Стихийное» программирование. Этап 2 - Структурный подход к программированию. Этап 3 - Объектный подход к программированию.
2. Основные этапы развития технологии разработки. Этап 4 - Компонентный подход и CASE-технологии. Этап 5 - Разработка, ориентированная на архитектуру и CASE-технологии.
3. Эволюция моделей жизненного цикла программного обеспечения. Каскадная модель. Спиральная модель.
4. Эволюция моделей жизненного цикла программного обеспечения. Макетирование. Быстрая разработка приложений.
5. Эволюция моделей жизненного цикла программного обеспечения. Компонентно-ориентированная модель. XP-процесс.
6. Стандарты, регламентирующие процесс разработки программного обеспечения.
7. Системный анализ, основные понятия. Системные ресурсы.
8. Анализ проблемы и моделирование предметной области с использованием системного подхода.
9. Пять этапов, которые необходимо осуществить, при анализе проблемы с использованием системного подхода.
10. Методология ARIS. Организационная модель.
11. Методология ARIS. Диаграмма цепочки добавленного качества.
12. Методология ARIS. Модели eEPC.
13. Стандарты IDEF0 – IDEF3.
14. Методология описания бизнес-процессов IDEF3.
15. Методология функционального моделирования IDEF0.

Уметь,

Владеть

16. Методы определения требований. Интервьюирование. «Мозговой штурм» и отбор идей.
17. Методы определения требований. Совместная разработка приложений (JAD – Joint Application Design).
18. Методы определения требований. Раскадровка. Обыгрывание ролей.
19. Методы определения требований. CRC-карточки (Class – Responsibility – Collaboration, класс – обязанность – взаимодействие). Быстрое прототипирование.
20. Формализация требований. Метод вариантов использования и его применение.
21. Формализация требований. Псевдокод. Конечные автоматы.
22. Формализация требований. Графические деревья решений. Диаграммы деятельности.
23. Техническое задание. Общие сведения. Назначение и цели создания (развития) системы. Характеристики объекта автоматизации.

24. Техническое задание. Требования к системе. Состав и содержание работ по созданию (развитию) системы.
25. Техническое задание. Порядок контроля и приемки системы. Требования к составу и содержанию работ по подготовке объекта автоматизации к вводу системы в действие. Требования к документированию. Источники разработки.
26. Планирование архитектуры.
27. Программный процесс и архитектурно-экономический цикл.
28. Суть программной архитектуры.
29. Проектирование архитектуры. Атрибутный метод проектирования.
30. Проектирование архитектуры. Создание макета системы.
31. Документирование программной архитектуры. Варианты применения архитектурной документации.
32. Документирование программной архитектуры. Документирование представления (view).
33. Методы анализа архитектуры. Метод анализа компромиссных архитектурных решений – комплексный подход к оценке архитектуры.
34. Методы анализа архитектуры. Метод анализа стоимости и эффективности – количественный подход к принятию архитектурно-проектных решений.
35. Использование архитектуры, управляемой моделью. Концепция архитектуры, управляемой моделью.
36. Использование архитектуры, управляемой моделью. Модельные точки зрения и модели MDA.
37. Язык объектных ограничений OCL. Типы данных и операции OCL. Инфиксная форма записи выражений OCL. Последовательности доступа к объектам в языке OCL.
38. Язык объектных ограничений OCL. Основные операции языка OCL.
39. Возможности технологии ECO.
40. Разработка приложений на основе ECO.

6. КРИТЕРИИ ОЦЕНИВАНИЯ КОМПЕТЕНЦИЙ

Оценка «отлично» выставляется студенту, если теоретическое содержание курса освоено полностью, без пробелов; исчерпывающе, последовательно, четко и логически стройно излагает материал; свободно справляется с задачами, вопросами и другими видами применения знаний; использует в ответе дополнительный материал все предусмотренные программой задания выполнены, качество их выполнения оценено числом баллов, близким к максимальному; анализирует полученные результаты; проявляет самостоятельность при выполнении заданий.

Оценка «хорошо» выставляется студенту, если теоретическое содержание курса освоено полностью, необходимые практические компетенции в основном сформированы, все предусмотренные программой обучения учебные задания выполнены, качество их выполнения достаточно высокое. Студент твердо знает материал, грамотно и по существу излагает его, не допуская существенных неточностей в ответе на вопрос.

Оценка «удовлетворительно» выставляется студенту, если теоретическое содержание курса освоено частично, но пробелы не носят существенного характера, большинство предусмотренных программой заданий выполнено, но в них имеются ошибки, при ответе на поставленный вопрос студент допускает неточности, недостаточно правильные формулировки, наблюдаются нарушения логической последовательности в изложении программного материала.

Оценка «неудовлетворительно» выставляется студенту, если он не знает значительной части программного материала, допускает существенные ошибки, неуверенно, с большими затруднениями выполняет практические работы, необходимые практические компетенции не сформированы, большинство предусмотренных программой обучения учебных заданий не выполнено, качество их выполнения оценено числом баллов, близким к минимальному.

7. УЧЕБНО-МЕТОДИЧЕСКОЕ И ИНФОРМАЦИОННОЕ ОБЕСПЕЧЕНИЕ ДИСЦИПЛИНЫ

7.1. Рекомендуемая литература

7.1.1. Основная литература:

1. Битюцкая, Н. И. Разработка программных приложений : лабораторный практикум / Н. И. Битюцкая. — Ставрополь : Северо-Кавказский федеральный университет, 2015. — 140 с.

2. Абрамов, Г. В. Проектирование и разработка информационных систем : учебное пособие для СПО / Г. В. Абрамов, И. Е. Медведкова, Л. А. Коробова. — Саратов : Профобразование, 2020. — 169 с. — ISBN 978-5-4488-0730-5. — Текст : электронный // Цифровой образовательный ресурс IPR SMART : [сайт]. — URL: https://www.iprbookshop.ru/88888.html (дата обращения: 10.04.2022). — Режим доступа: для авторизир. пользователей. - DOI: https://doi.org/10.23682/88888

7.1.2. Перечень дополнительной литературы:

1. Баженова, И. Ю. Основы проектирования приложений баз данных : учебное пособие / И. Ю. Баженова. — 3-е изд. — Москва : Интернет-Университет Информационных Технологий (ИНТУИТ), Ай Пи Ар Медиа, 2020. — 324 с. — ISBN 978-5-4497-0682-9. — Текст : электронный // Цифровой образовательный ресурс IPR SMART : [сайт]. — URL: https://www.iprbookshop.ru/97569.html (дата обращения: 10.04.2022). — Режим доступа: для авторизир. пользователей

2. Гвозденко, Н. П. Разработка блок-схем алгоритмов : учебное пособие / Н. П. Гвозденко, С. А. Суслова. — Липецк : Липецкий государственный технический университет, ЭБС АСВ, 2021. — 59 с. — ISBN 978-5-00175-055-0. — Текст : электронный // Цифровой образовательный ресурс IPR SMART : [сайт]. — URL: https://www.iprbookshop.ru/116169.html (дата обращения: 10.04.2022). — Режим доступа: для авторизир. Пользователей

3. Савельев, А. О. Проектирование и разработка веб-приложений на основе технологий Microsoft : учебное пособие / А. О. Савельев, А. А. Алексеев. — 4-е изд. — Москва : Интернет-Университет Информационных Технологий (ИНТУИТ), Ай Пи Ар Медиа, 2022. — 418 с. — ISBN 978-5-4497-1650-7. — Текст : электронный // Цифровой образовательный ресурс IPR SMART : [сайт]. — URL: https://www.iprbookshop.ru/120486.html (дата обращения: 10.04.2022). — Режим доступа: для авторизир. пользователей

7.2. Перечень учебно-методического обеспечения самостоятельной работы обучающихся по дисциплине (модулю)

1. Методические рекомендации по выполнению лабораторных работ по дисциплине " Технология разработки программного обеспечения "

2. Методические рекомендации по организации самостоятельной работы студентов по дисциплине " Технология разработки программного обеспечения "

1. Методические указания по выполнению курсовой работы по дисциплине «Технология разработки программного обеспечения».

7.3. Перечень ресурсов информационно-телекоммуникационной сети «Интернет», необходимых для освоения дисциплины (модуля)

1. http://el.ncfu.ru/ – система управления обучением ФГАОУ ВО СКФУ. Дистанционная поддержка дисциплины «Цифровая грамотность и обработка данных»
2. http://www.un.org - Сайт ООН Информационно-коммуникационные технологии
3. http://www.intuit.ru – Интернет-Университет Компьютерных технологий.

8. Перечень информационных технологий, используемых при осуществлении образовательного процесса по дисциплине (модулю), включая перечень программного обеспечения и информационных справочных систем

При чтении лекций используется компьютерная техника, демонстрации презентационных мультимедийных материалов. На семинарских и практических занятиях студенты представляют презентации, подготовленные ими в часы самостоятельной работы.

Информационные справочные системы:

Информационно-справочные и информационно-правовые системы, используемые при изучении дисциплины:

1	КонсультантПлюс - http://www.consultant.ru/

Программное обеспечение:

1	Операционная система: Microsoft Windows 8: 2013-02(3000). Бессрочная лицензия. Договор № 01-эа/13 от 25.02.2013. Окончание бесплатной поддержки – 2024-01 ИЛИ Операционная система: Microsoft Windows 10: 2016-08(20), 2017-10(67), 2018-01(18), 2018-04(6), 2018-05(6), 2019-02(7). Бессрочная лицензия. Договоры № 27-эа/16 от 02.08.2016. и № 0321100021117000009_229123 от 10.10.2017. На текущий момент окончания поддержки не анонсировано.
2	Базовый пакет программ Microsoft Office (Word, Excel, PowerPoint). MicrosoftOfficeStandard 2013: договор № 01-эа/13 от 25.02.2013г., Лицензирование Microsoft Office https://support.microsoft.com/ru-ru/lifecycle/search/16674 Дата начала жизненного цикла 09.01.2013г.; набор обновлений Office 2013 Service Pack1 Дата начала жизненного цикла 25.02.2014г., Дата окончания основной фазы поддержки 10.04.2018; Дополнительная дата окончания поддержки 11.04.2024г.