МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Пермский государственный национальный исследовательский университет»

Колледж профессионального образования

ТЕХНОЛОГИЯ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Методические рекомендации

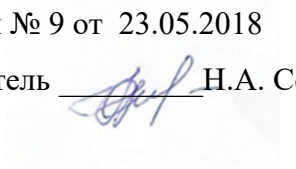
для лабораторных работ по изучению дисциплины

для студентов Колледжа профессионального образования

специальности

09.02.03 Программирование в компьютерных системах

Утверждено на заседании ПЦК
Информационных технологий
Протокол № 9 от 23.05.2018
председатель _____Н.А. Серебрякова



Пермь 2018

Составитель:

Бочкарев Алексей Михайлович, преподаватель первой квалификационной категории, преподаватель ПГНИУ

ТЕХНОЛОГИЯ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ : методические указания по практической работе для студентов Колледжа профессионального образования по специальностям 09.02.03 Программирование в компьютерных системах/ сост. А.М. Бочкарев; Колледж проф. образ. ПГНИУ. – Пермь, 2020. – 16 с.

Методические указания «ТЕХНОЛОГИЯ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ » разработаны на основе требований Федерального государственного образовательного стандарта среднего профессионального образования по специальностям 09.02.03 Программирование в компьютерных системах и 09.02.04 Информационные системы (по отраслям) для оказания помощи студентам специальностей 09.02.03 Программирование в компьютерных системах по дисциплине «ТЕХНОЛОГИЯ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ». Содержат типичные лабораторные задания по всем разделам дисциплины.

Предназначены для студентов Колледжа профессионального образования ПГНИУ специальностей 09.02.03 Программирование в компьютерных системах (СПО) всех форм обучения.

Печатается по решению педагогического совета Колледжа профессионального образования Пермского государственного национального исследовательского университета

СОДЕРЖАНИЕ

Введение

УВАЖАЕМЫЙ СТУДЕНТ!

Методические указания по **МДК 03.01 Технология разработки программного обеспечения** для выполнения практических работ созданы Вам в помощь для работы на занятиях, подготовки к практическим работам, правильного составления отчетов.

Приступая к выполнению практической работы, Вы должны внимательно прочитать цель и задачи занятия, ознакомиться с требованиями к уровню Вашей подготовки в соответствии с федеральными государственными стандартами третьего поколения (ФГОС-3), краткими теоретическими и учебно-методическими материалами по теме практической работы, ответить на вопросы для закрепления теоретического материала.

Все задания к практической работе Вы должны выполнять в соответствии с инструкцией, анализировать полученные в ходе занятия результаты по приведенной методике.

Отчет о практической работе Вы должны выполнить по приведенному алгоритму, опираясь на образец.

Наличие положительной оценки по практическим работам необходимо для получения зачета по МДК 03.01 Технология разработки программного обеспечения и допуска к экзамену, поэтому в случае отсутствия на уроке по любой причине или получения неудовлетворительной оценки за практическую работу Вы должны найти время для ее выполнения или пересдачи.

Внимание! Если в процессе подготовки к практическим работам или при решении задач у Вас возникают вопросы, разрешить которые самостоятельно не удается, необходимо обратиться к преподавателю для получения разъяснений или указаний в дни проведения дополнительных занятий.

Время проведения дополнительных занятий можно узнать у преподавателя или посмотреть на двери его кабинета.

Желаем Вам успехов!!!

Пояснительная записка

Методические указания по выполнению практических работ студентами очной формы обучения по МДК 03.01 Технология разработки программного обеспечения предназначены для реализации образовательных результатов, заявленных во ФГОС третьего поколения по специальности 09.02.03 Программирование в компьютерных системах.

Перечень практических работ в соответствии с указанием разделов и тем:

Раздел 2 Основные процессы разработки программного обеспечения

Тема 2.3 Планирование работ по созданию программного продукта

ПР №1 **Построение диаграмм Ганта при планировании небольшого проекта**

Раздел 3 Проектирование и программирование программных продуктов

Тема 3.2 Разработка программных модулей

ПР №2 Разработка документации. Стадии «Техническое задание»

ПР №3 Разработка документации. Стадия «Эскизный проект»

ПР№4 Разработка документации. Стадия «Технический проект»

ПР№5 Применение методов объектно-ориентированного проектирования

Раздел 4 Отладка, тестирование и сопровождение программных продуктов

Тема 4.1 Тестирование программного продукта

ПР№6 Тестирование программного продукта методом «белого ящика»

ПР№7 Тестирование программного продукта методом «черного ящика»

ПР№8 Автоматизированное тестирование

Тема 4.2 Отладка программных продуктов

ПР №9 **Отладка программного продукта**

Раздел 5 Методы организации работы в коллективах разработчиков программного обеспечения

Тема 5.1 Технология коллективной разработки ПО

ПР№ 10 Коллективная разработка программного продукта

Образовательные результаты, заявленные во ФГОС третьего поколения:

Студент должен

уметь:

- владеть основными методологиями процессов разработки программного обеспечения;

- использовать методы для получения кода с заданной функциональностью и степенью качества;

знать:

- модели процесса разработки программного обеспечения;
- основные принципы процесса разработки программного обеспечения;
- основные подходы к интегрированию программных модулей;
- основные методы и средства эффективной разработки;
- основы верификации и аттестации программного обеспечения;
- концепции и реализации программных процессов;
- принципы построения, структуры и приемы работы с инструментальными средствами, поддерживающими создание программного обеспечения;
- методы организации работы в коллективах разработчиков программного обеспечения;
- основные положения метрологии программных продуктов, принципы построения, проектирования и использования средств для измерений характеристик и параметров программ, программных систем и комплексов;
- стандарты качества программного обеспечения;
- методы и средства разработки программной документации

Задачи практической работы:

1. Повторить (изучить) теоретический материал.
2. Выполнить практическую работу.
3. Оформить и сдать отчет по практической работе.
4. По окончании всего цикла практических работ предоставить папку-отчет на бумажном носителе.

Обеспеченность занятия:

1. Учебно-методическая литература:

1. В. В. Бахтизин, Л. А. Глухова. Технология разработки программного обеспечения : учеб. пособие / – Минск : БГУИР, 2010
2. К.Вигерс. Разработка требований к программному обеспечению. Издательско-торговый дом «Русская редакция», 2009
3. Гагарина Л.Г., Кокорева Е.В., Виснадул Б.Д. Технология разработки программного обеспечения: учебное пособие.- М.: ИД «Форум»: ИНФРА-М, 2009
4. А.В. Рудаков .Технология разработки программных продуктов. М.: ACADEMIA, 2012
5. Рудаков А.В., Федорова Г.Н. Технология разработки программных продуктов. Практикум: -М.:ACADEMA, 2010

2. Технические средства обучения:

- компьютер преподавателя с лицензионным программным обеспечением и мультимедиа проектор.

- компьютеры студентов с лицензионным программным обеспечением, объединенные в локальную вычислительную сеть;

- принтер, сканер.

3. Программное обеспечение: MS Windows 7, MS Office 2010, Erwin Data Modeler

4. Раздаточные материалы:

- файлы с методическими указаниями по выполнению практической работы, инструкционные карты с заданиями;

- файл с шаблоном отчета

Инструкция по выполнению практической работы

1. Внимательно прочитать наименование и цель практической работы.

2. Изучить краткие теоретические сведения

3. Ответить на контрольные вопросы для закрепления теоретического материала, используя шаблон отчета в программе MS Word.

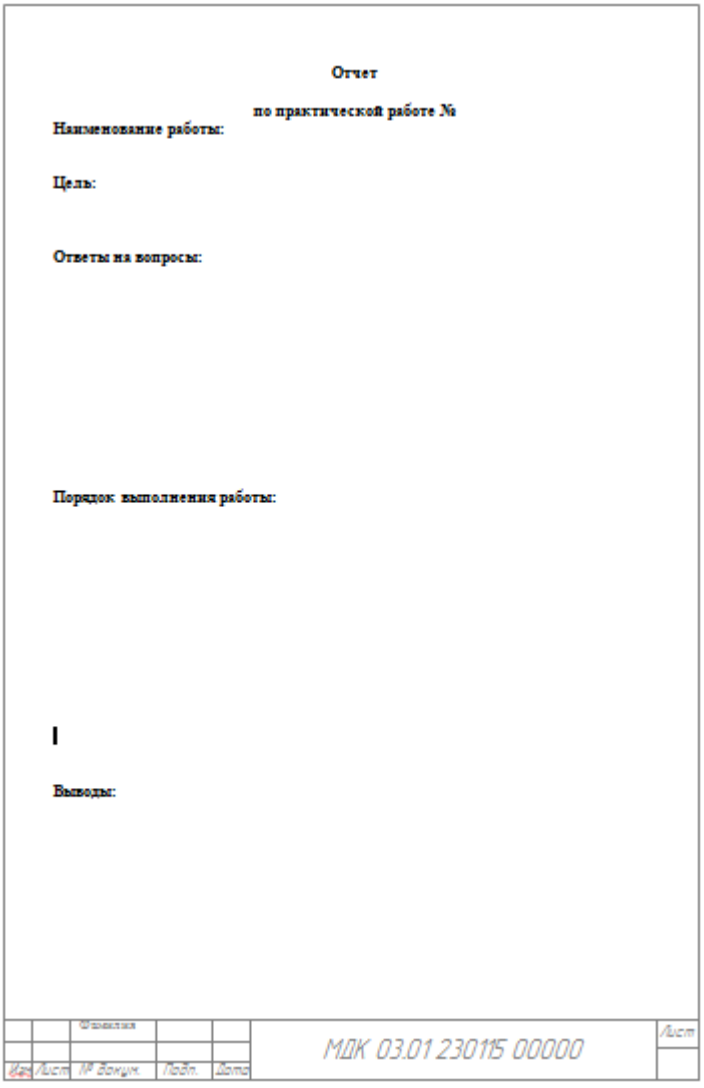
4. Выполнить задания, в соответствии с указаниями.

5. В заданиях, где используется программное обеспечение, сохранить результаты работы на диске E: в папке СТУДЕНТЫ – папка ГРУППА. В папке ГРУППА создать свою папку с именем вашей фамилии, например ИВАНОВ.

Образец отчета по практической работе

В методических указаниях по выполнению самостоятельной работы по **МДК 03.01 Технология разработки программного обеспечения** указаны порядок и требования по оформлению отчета.

Защита отчета по практической работе заключается в предъявлении преподавателю полученных результатов, демонстрации полученных навыков и ответах на вопросы преподавателя.



Отчет

по практической работе №

Наименование работы:

Цель:

Ответы на вопросы:

Порядок выполнения работы:

Выводы:

МДК 03.01 230115 00000

Образец заполнения отчета

Практическая работа № 1

Построение Gantt-диаграммы при планировании небольшого проекта

Цель работы состоит в приобретении навыков визуализации информации с помощью диаграмм; расширения навыков построения нестандартных диаграмм в среде электронных таблиц применении средств деловой графики.

Краткие теоретические и учебно-методические материалы по теме практической работы

Диаграммы Ганта (англ.Gantt chart, также ленточная диаграмма) - это популярный тип столбчатых диаграмм, который используется для иллюстрации плана, графика работ по какому-либо проекту. Является одним из методов планирования проектов, представляет собой изображение календарного графика задач в проекте. График Ганта является своеобразным стандартом в области управления проектами, ведь именно с его помощью появляется возможность показать структуру выполнения всех этапов проекта наглядно.

Диаграммы Ганта позволяет:

- визуально оценить последовательность задач, их относительную длительность и протяженность проекта в целом;

- сравнить планируемый и реальный ход выполнения задач;

- детально проанализировать реальный ход выполнения задач.

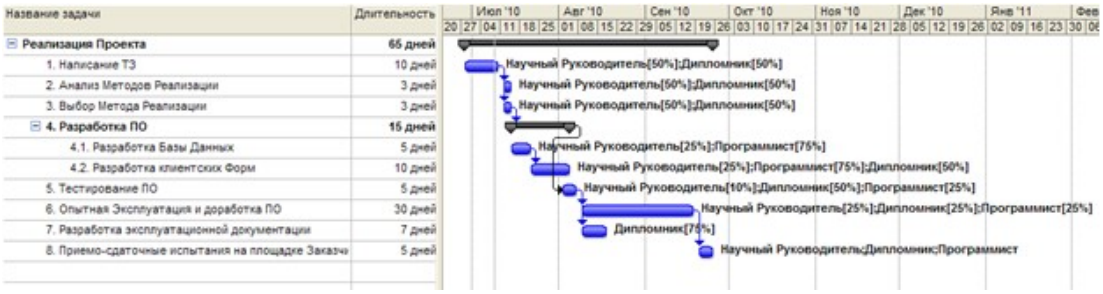
Диаграмма даёт возможность решить одну из основных задач и показать персоналу, над чем следует работать, какие ресурсы применять в процессе и с какой скоростью выполнять те или иные задачи. Вся информация подаётся в сжатом виде, без использования запутанных таблиц и огромного количества текста. При этом суть ясна и понятна всем, без исключения, участникам проекта.

Использование диаграммы значительно упрощает управление проектами небольших масштабов и даёт возможность всегда держать деятельность сотрудников под контролем.

Диаграмма Ганта состоит из полос, ориентированных вдоль оси времени. Каждая полоса на диаграмме представляет отдельную задачу в составе проекта (вид работы), её концы - моменты начала и завершения работы, её протяженность - длительность работы. Вертикальной осью диаграммы служит перечень задач. Кроме того, на диаграмме могут быть отмечены совокупные задачи, проценты завершения, указатели последовательности и зависимости работ, метки ключевых моментов (вехи), метка текущего момента времени «Сегодня» и др.

Диаграмма может использоваться для представления текущего состояния выполнения работ: часть прямоугольника, отвечающего задаче, заштриховывается, отмечая процент выполнения задачи; показывается вертикальная линия, отвечающая моменту «сегодня».

Часто диаграмма Ганта соседствует с таблицей со списком работ, строки которой соответствуют отдельно взятой задаче, отображенной на диаграмме, а столбцы содержат дополнительную информацию о задаче. Пример такой таблицы представлен ниже.



В электронном виде диаграмма Ганта строится с помощью таких средств, как: MS Project, MS Visio, Primavera и прочее.

Разумеется, существует множество других, более совершенных программ, облегчающих управление проектами. Диаграмма Ганта любой сложности может быть легко построена с помощью таких приложений, как: SchedRoll; Gantt Designer; Mindjet JCVGantt Pro; и многих других.

Кроме того, существует целый ряд онлайн-сервисов, которые предоставляют своим пользователям возможность не только планировать свои дела, но и получать регулярные отчёты, уведомления о текущем статусе задач по электронной почте.

Рассмотрим основные принципы построения диаграммы:

Графически задачи отображаются сверху вниз слево направо.

Задача может состоять из нескольких подзадач, но не менее 2-х (на графике Задача 4. Разработка ПО и ее подзадачи 4.1. и 4.2.).

Задачи могут выполняться последовательно (Задачи 1. и 2.) и параллельно (Задачи 2. и 3.).

Загруженность исполнителей ставится с учетом выполнения работ во времени (Обратите внимание на Задачи 2. и 3.).

Этапы календарного планирования

1) Определение основных этапов проекта. Как правило, выделяется минимум три этапа проекта. Этапы можно обозначить, руководствуясь следующими принципами:

a. По времени – делить на примерно равные временные промежутки;

b. По характеру работ – делить проект на различные блоки работ, отличные друг от друга по содержанию и характеру;

c. По результатам – выделить значимые, измеряемые результаты.

2) Декомпозиция этапов на меньшие и конкретные работы (задачи). Если возможно, то максимально подробно, если нет – руководствуйтесь принципами из пункта 1, то есть, разделите каждый этап как минимум еще на 3 задачи.

3) Определение последовательности работ.

4) Определение логических связей. Могут быть ограничения, например, невозможно начать выполнять задачу, не закончив предыдущую, либо не закончив три предыдущих задачи одного блока.

5) Спланировать сроки выполнения задач с построением диаграммы Ганта, диаграмма будет рассмотрена ниже.

6) Определение потребности в ресурсах:

a. людские ресурсы - определить ответственных по каждой задаче;

b. машины и механизмы;

c. помещения;

d. материалы и так далее.

7) Определение стоимости ресурсов и трудозатрат.

8) Оптимизация диаграммы Ганта с учетом загрузки ресурсов. Например, первоначально вы не знали, кого назначите выполнять две параллельные задачи, а потом оказалось, что их может выполнить одно должностное лицо и никак иначе, соответственно, эти задачи станут последовательными.

Оптимизация диаграммы Ганта:

1) Рассмотрите возможность выполнения задач параллельно.

2) Определите крайние точки проекта и отдельных задач.

3) Установите связи для последовательных задач.

4) Задачи, которые можно сделать позже передвиньте в конец.

5) Определите критичный путь – последовательность задач, которая определяет длительность проекта, и рассмотрите каждую на возможность сокращения по времени.

Основные достоинства и недостатки описанного метода планирования и управления.

Главным преимуществом является графическая подача материала. Удобство работы с графиками Ганта – возможность чётко выделить и обозначить этапы работы над проектом и одновременное отображение мероприятий и сроков их выполнения. За счёт представления заданий в виде различных цветных полос все члены команды могут буквально с первого взгляда определить свои задачи.

Диаграммы Ганта являются отличным презентационным инструментом, который способен продемонстрировать ключевые приоритеты проекта. То есть, как только руководящие лица выделяют и распределяют каждый из имеющихся в наличии ресурсов, команда моментально узнаёт об этом и следует дальнейшим указаниям. Данное свойство графика Ганта крайне полезно для руководителей высшего звена – используя его, можно намного проще подготовить подробный, ёмкий отчёт о состоянии выполнения различных проектов.

Тем не менее, как и у любого другого метода планирования, у диаграммы Ганта есть свои недостатки.

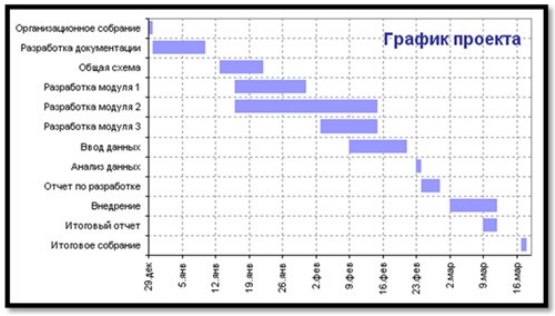
Ключевым понятием диаграммы Ганта является «Веха» - метка значимого момента в ходе выполнения работ, общая граница двух или более задач. Вехи позволяют наглядно отобразить необходимость синхронизации, последовательности в выполнении различных работ. Вехи, как и другие границы на диаграмме, не являются календарными датами. Сдвиг вехи приводит к сдвигу всего проекта. Поэтому диаграмма Ганта не является, строго говоря, графиком работ. И это один из основных её недостатков. Кроме того, диаграмма Ганта не отображает значимости или ресурсоемкости работ, не отображает сущности работ (области действия). Для крупных проектов диаграмма Ганта становится чрезмерно тяжеловесной и теряет всякую наглядность.

Недостатком является зависимость задач. Довольно часто в процессе презентации проектов у руководителей возникает необходимость показать, какие из указанных заданий связаны друг с другом. Но, к сожалению, сам формат диаграммы не позволяет сделать этого. Для того чтобы обойти это ограничение, менеджеры прибегают к различным хитростям: например, добавляют в график специальные вертикальные линии, которые демонстрируют ключевые зависимости. Однако это лишь временное решение, не способное передать информацию в полном объёме.

Ещё одним минусом графиков Ганта можно назвать их негибкость. В наши дни проекты не являются статичными – в них постоянно происходят какие-то изменения, сдвиги, учесть которые в диаграмме просто невозможно. Прежде чем приступать к построению графика, руководителям приходится просчитывать всё до мелочей, ведь при малейшем изменении оценки нужно перерисовывать «с нуля» всю диаграмму. И это не говоря уже о том, что возможность проиллюстрировать несколько разных способов планирования за один раз также отсутствует.

Вне зависимости от того, зачем вам нужна диаграмма Ганта, программа (даже самая «продвинутая») не сможет отобразить значимость и ресурсоёмкость тех или иных работ, их сущность. А потому для особо масштабных проектов она используется крайне редко.

Указанные выше недостатки и ограничения серьёзно ограничивают область применения диаграммы. Тем не менее, в настоящее время диаграмма Ганта является стандартом де-факто в теории и практике управления проектами. В практике управления проектами данный метод чрезвычайно распространён.



Календарный план, план-график или диаграмма Ганта после построения становится реальным управленческим инструментом, во-первых, возможно видеть весь проект в виде одной схемы взаимоувязанных задач и не нужно держать в голове, во-вторых, на этом же графике вы отмечаете выполнение задач и видите отклонение от графика.

Задания для практического занятия:

1. Выполнить упражнения. Сохранить результаты на диске Е: в папке СТУДЕНТЫ.

2. Ответить на контрольные вопросы. Оформить в отчет.

Упражнения

Задание 1. Построение диаграммы Ганта. Стрелочная диаграмма.

1.Нарисуйте таблицу, в левый столбец которой занесите наименования выполняемых мероприятий. Наименования мероприятий следует расставлять сверху вниз в порядке их выполнения.

2.Выберите удобную периодичность контроля над выполнением занесенных в таблицу мероприятий и проставьте ее в верхней строке нарисованной таблицы. В качестве периодичности выполнения работ могут выступать недели, месяцы, кварталы и т.д.

3.В строке каждого мероприятия следует нарисовать стрелку, которая начинается в столбце запланированного срока начала выполнения этого мероприятия, а заканчивается в столбце запланированного срока завершения выполнения рассматриваемого мероприятия.

Пример диаграммы Ганта:

Задание 2.

1. Сбор данных.

Для того чтобы построить график, понадобятся следующие данные:

- координаты всех наборов данных (откуда должен начинаться каждый из столбиков);
- название каждого этапа;
- продолжительность каждого этапа.

Для удобства вписываем их сразу в соответствующие поля таблицы. После того как Вы ввели всю требуемую информацию, можно переходить к созданию самой диаграммы.

Важно: проследите за тем, чтобы все форматы данных были указаны правильно: в частности, это касается дат.

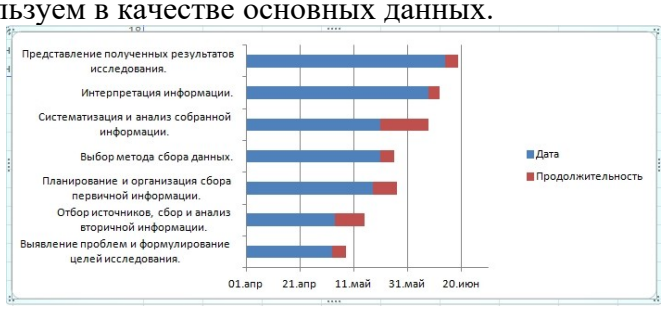
Особенность диаграммы Ганта в том, что ее столбики начинаются в произвольном месте, в то время как наиболее похожая на нее диаграмма – линейчатая диаграмма начинается с оси. Поэтому мы используем один из самых распространенных трюков с диаграммами – скроем какие-то данные.

Создадим таблицу.

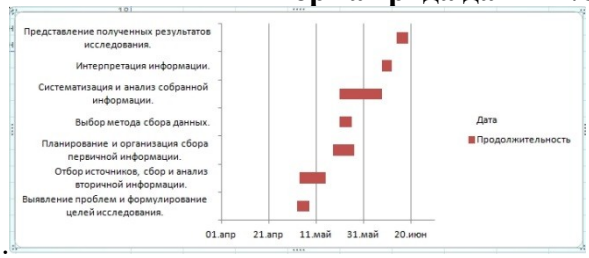
	A	B	C
1		Дата	Продолжительность
2	Выявление проблем и формулирование целей исследования.	03.май	5
3	Отбор источников, сбор и анализ вторичной информации.	04.май	11
4	Планирование и организация сбора первичной информации.	18.май	9
5	Выбор метода сбора данных.	21.май	5
6	Систематизация и анализ собранной информации.	21.май	18
7	Интерпретация информации.	08.июн	4
8	Представление полученных результатов исследования.	14.июн	5
9			

Обратите внимание, что не озаглавлены названия этапов, т.е. ячейка A1 пустая. Этот прием позволяет с первого раза в диаграмме определить, где находятся данные, а где подписи к ним.

Используем линейчатую диаграмму. Надо брать ее с накоплением, так как нужен второй ряд данных, который и используем в качестве основных данных.



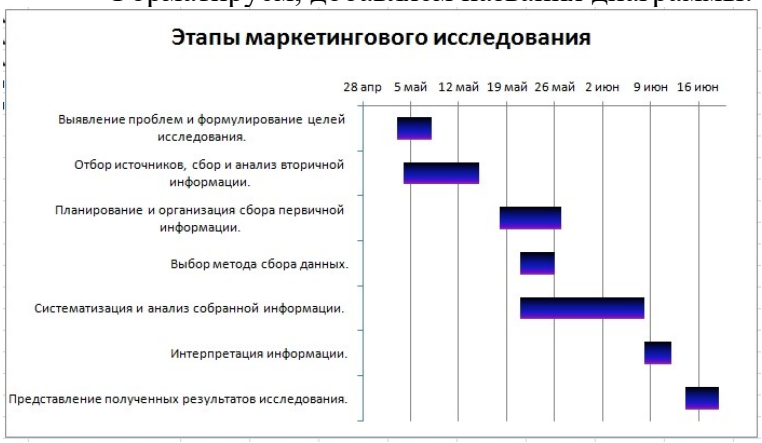
Второй необходимый шаг – это скрытие первого ряда. Для этого делаем его невидимым. Щелкаем на синих данных, правой кнопкой мыши **Формат ряда данных/Заливка/Нет Заливки**



По умолчанию данные расположены в порядке снизу вверх. Исправляем это положение - кликаем на ось категорий (подписей) правой кнопкой мыши - **Формат оси/Параметры оси/ Обратный порядок категорий**. Все, основная диаграмма готова, нужно теперь сделать еще кое-какое полезное форматирование:

- **Убираем легенду.** Она просто уже неактуальна для диаграммы Ганта. Выделяем и кнопкой Delete удаляем.
- **Определяем границы.** Excel сам определяет откуда начинаются значения, т.е. на рисунке сверху шкала начинается с первого апреля. Необходимо установить другую дату. Поэтому правой кнопкой мыши на оси дат **Формат оси/Параметры оси/Максимальное и минимальное значения**. На диаграмме это 3 мая (точнее, 28 апреля, понедельник) и 20 июня. Правда, есть нюанс, в Excel 2007 надо ставить числа в числовом формате.
- **Ставим недельные шкалы**. Там же в параметрах оси, ставим 7 в окне **цена основных делений**. Вот для чего потребовалось, чтобы первая дата стояла понедельником.

Форматируем, добавляем названия диаграммы:



Задание 3. Создание диаграммы Ганта в MS Visio

При помощи диаграммы Ганта можно составить расписание задач проекта, а затем отследить его ход.

1.В меню Файл последовательно выберите команды Создать, Расписание, а затем – команду Диаграмма Ганта.

2.На вкладке Дата введите количество задач для начала работы, единицы времени, которые нужно отобразить, и диапазон дат для проекта.

ПРИМЕЧАНИЕ. Параметры форматирования можно изменить в любой момент. В меню Диаграмма Ганта выберите команду Параметры, а затем перейдите на вкладку Формат.

3.Замените имена задач по умолчанию именами задач, соответствующими проекту, а также замените даты начала и окончания задач и длительность.

4.В диаграмме Ганта в столбце Название задачи выделите ячейку, содержащую задачу, которую необходимо переименовать, а затем введите новое имя.

5.В диаграмме Ганта выделите ячейку, содержащую дату, которую необходимо изменить, а затем введите новую дату.

ПРИМЕЧАНИЕ. Дату для итоговой задачи изменить нельзя. Даты итоговых задач изменяются только при изменении даты одной или нескольких задач, расположенных уровнем ниже итоговой задачи.

6.В диаграмме Ганта выделите ячейку, содержащую длительность, которую нужно изменить, а затем введите новое значение длительности. Используйте следующие сокращения: м - минуты, ч - часы, д - дни и н - недели.

ПРИМЕЧАНИЕ. Между числом и сокращением не должно быть пробела. Например, для указания длительности в 5 дней введите 5д.

СОВЕТ. Длительность можно также изменить, выделив область задач и перетащив маркер выделения ▪ с правой стороны области задач на новую дату окончания на временной шкале.

7.Добавьте дополнительные задачи в диаграмму Ганта.

- Выделите рамку проекта, щелкнув сплошную линию вокруг диаграммы Ганта.

- Перетащите вниз маркер выделения ▪, расположенный по центру нижней части рамки. Будут созданы новые строки задач, заполняющие пространство.

- Выделите ячейку Название задачи одной из новых задач, а затем введите имя задачи.

СОВЕТ. Положение задач в диаграмме Ганта можно изменить, перетащив строки задач внутри рамки диаграммы.

10.Добавьте вехи в диаграмму Ганта.

- Из набора элементов Фигуры диаграммы Ганта перетащите на страницу документа фигуру Веха и поместите ее между задачами, которые предваряет или за которыми следует веха.

ПРИМЕЧАНИЕ. При добавлении фигуры Веха в диаграмму Ганта автоматически добавляется строка со значением длительности, равным 0 (нулю).

- Введите имя и дату для вехи.

СОВЕТ. Задачу в вехе можно изменить, задав значение ее длительности равным 0. Подобным образом можно изменить веху в задаче, задав положительное значение длительности.

11.Создайте зависимости между задачами в диаграмме Ганта.

- Сначала выделите область задач или веху, из которых нужно установить связь, а затем нажмите клавишу SHIFT и выделите зависимую задачу или веху.

- Щелкните правой кнопкой мыши одну из фигур, а затем в контекстном меню выберите команду Связать задачи.

Задание 4. Имитация диаграммы Ганта

Этапы имитации диаграммы Ганта:

1. Для имитации диаграммы Ганта необходимо ввести данные на листе рабочей книги Microsoft Office Excel (например, в соответствии с таблицей 3)

	A	B	C
1	Мероприятия	Начало, день	Длительность, дней
2	Мероприятие 1	0	2
3	Мероприятие 2	2	3
4	Мероприятие 3	2	5
5	Мероприятие 4	4	2
6	Мероприятие 5	5	3
7	Мероприятие 6	8	2
8			

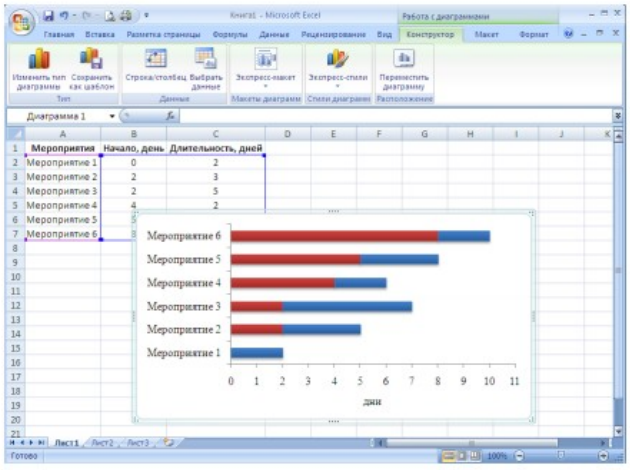
Таблица 3 – Исходные данные для построения диаграммы Ганта

№	Мероприятия	Начало, день	Длительность, дней
1	Мероприятие 1	0	2
2	Мероприятие 2	2	3
3	Мероприятие 3	2	5
4	Мероприятие 4	4	2
5	Мероприятие 5	5	3
6	Мероприятие 6	8	2

2. Выберите данные, которые нужно показать на диаграмме Ганта.

3. На вкладке Вставка в группе Диаграммы щелкните Линейчатая.

4. В группе Плоская линейчатая диаграмма выберите Линейчатая диаграмма с накоплением.



5. Щелкните на диаграмме область диаграммы, при этом появится панель Работа с диаграммами с вкладками Конструктор, Макет и Формат.

6. Выберите на диаграмме значения интервалов относящие к группе значений первого столбца, а далее измените его Заливку выберите вариант Нет заливки.

7. Таким образом, получаем имитацию диаграммы Ганта.

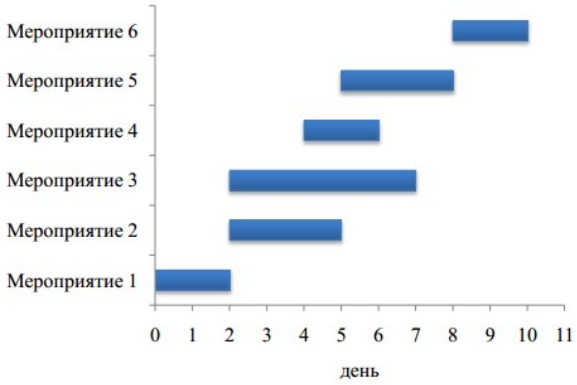


Диаграмма реализации мероприятий

Вопросы для закрепления теоретического материала к практическому занятию:

1. Для чего нужна диаграмма Ганта?
2. История появления первого графика
3. Диаграмма Ганта в современном мире
4. Другие программы для создания графика
5. основные принципы построения диаграммы
6. Преимущества и недостатки метода
7. Что такое календарное планирование
8. Где применимо календарное планирование

Порядок выполнения отчета по практической работе

1. Изучить теоретический материал. Ответить на контрольные вопросы.
2. Оформить работу в соответствии с шаблоном (Приложение 1). При оформлении использовать MS Office.
3. Сохранить результаты работы в соответствующей папке на диске E:.

Практическая работа №2

Разработка документации. Стадия «Техническое задание»

Цель работы состоит в приобретении навыков разработки технического задания на программный продукт, ознакомиться с правилами написания технического задания

Краткие теоретические и учебно-методические материалы по теме практической работы

Техни́ческое зада́ние (**ТЗ**, **техзада́ние**) - исходный документ для проектирования сооружения или промышленного комплекса, конструирования технического устройства (прибора, машины, системы управления и т. д.), разработки информационных систем, стандартов либо проведения научно-исследовательских работ (НИР).

ТЗ содержит основные технические требования, предъявляемые к сооружению, изделию или услуге и исходные данные для разработки. В ТЗ указываются назначение объекта, область его применения, стадии разработки конструкторской (проектной, технологической, программной и т.п.) документации, её состав, сроки исполнения и т. д., а также особые требования, обусловленные спецификой самого объекта либо условиями его эксплуатации. Как правило, ТЗ

составляют на основе анализа результатов предварительных исследований, расчётов и моделирования.

Типовые требования к составу и содержанию технического задания приведены в таблице 1.

Таблица1 - Состав и содержание технического задания (ГОСТ 34.602- 89)

№ пп	Раздел	Содержание
1	Общие сведения	- полное наименование системы и ее условное обозначение - шифр темы или шифр (номер) договора; - наименование предприятий разработчика и заказчика системы, их реквизиты - перечень документов, на основании которых создается ИС - плановые сроки начала и окончания работ - сведения об источниках и порядке финансирования работ - порядок оформления и предъявления заказчику результатов работ по созданию системы, ее частей и отдельных средств
2	Назначение и цели создания (развития) системы	- вид автоматизируемой деятельности - перечень объектов, на которых предполагается использование системы - наименования и требуемые значения технических, технологических, производственно-экономических и др. показателей объекта, которые должны быть достигнуты при внедрении ИС
3	Характеристика объектов автоматизации	- краткие сведения об объекте автоматизации - сведения об условиях эксплуатации и характеристиках окружающей среды
4	Требования к системе	Требования к системе в целом: - требования к структуре и функционированию системы (перечень подсистем, уровни иерархии, степень централизации, способы информационного обмена, режимы функционирования, взаимодействие со смежными системами, перспективы развития системы) - требования к персоналу (численность пользователей, квалификация, режим работы, порядок подготовки) - показатели назначения (степень приспособляемости системы к изменениям процессов управления и значений параметров) - требования к надежности, безопасности, эргономике, транспортабельности, эксплуатации, техническому обслуживанию и ремонту, защите и сохранности информации, защите от внешних воздействий, к патентной чистоте, по стандартизации и унификации Требования к функциям (по подсистемам) : - перечень подлежащих автоматизации задач - временной регламент реализации каждой функции - требования к качеству реализации каждой функции, к форме представления выходной информации, характеристики точности, достоверности выдачи результатов - перечень и критерии отказов Требования к видам обеспечения: - математическому (состав и область применения мат. моделей и методов, типовых и разрабатываемых алгоритмов) - информационному (состав, структура и организация данных, обмен данными между компонентами системы, информационная совместимость со смежными системами, используемые классификаторы, СУБД, контроль данных и ведение информационных массивов, процедуры придания юридической силы выходным документам) - лингвистическому (языки программирования, языки взаимодействия пользователей с системой, системы кодирования, языки ввода- вывода) - программному (независимость программных средств от платформы, качество программных средств и способы его контроля, использование фондов алгоритмов и программ) - техническому

		- метрологическому - организационному (структура и функции эксплуатирующих подразделений, защита от ошибочных действий персонала) - методическому (состав нормативно- технической документации
5	Состав и содержание работ по созданию системы	- перечень стадий и этапов работ - сроки исполнения - состав организаций — исполнителей работ - вид и порядок экспертизы технической документации - программа обеспечения надежности - программа метрологического обеспечения
6	Порядок контроля и приемки системы	- виды, состав, объем и методы испытаний системы - общие требования к приемке работ по стадиям - статус приемной комиссии
7	Требования к составу и содержанию работ по подготовке объекта автоматизации к вводу системы в действие	- преобразование входной информации к машиночитаемому виду - изменения в объекте автоматизации - сроки и порядок комплектования и обучения персонала
8	Требования к документированию	- перечень подлежащих разработке документов - перечень документов на машинных носителях
9	Источники разработки	- документы и информационные материалы, на основании которых разрабатывается ТЗ и система

Порядок разработки технического задания

Разработка технического задания выполняется в следующей последовательности. Прежде всего, устанавливают набор выполняемых функций, а также перечень и характеристики исходных данных.

Затем определяют перечень результатов, их характеристики и способы представления.

Далее уточняют среду функционирования программною обеспечения: конкретную комплектацию и параметры технических средств, версию используемой операционной системы и, возможно, версии и параметры другого установленного программного обеспечения, с которым предстоит взаимодействовать будущему программному продукту.

В случаях, когда разрабатываемое программное обеспеченно собирает и хранит некоторую информацию или включается в управление каким-либо техническим процессом, необходимо также четко регламентировать действия программы в случае сбоев оборудования и энергоснабжения.

1. Общие положения

1.1 Техническое задание оформляют в соответствии с ГОСТ 19.106-78 на листах формата А4 и А3 по ГОСТ 2.301-68, как правило, без заполнения полей листа. Номера листов (страниц) проставляют в верхней части листа над текстом.

1.2 Лист утверждения и титульный лист оформляют в соответствии с ГОСТ 19.104-78. Информационную часть (аннотацию и содержание), лист регистрации изменений допускается и в документ не включать.

1.3 Для внесения изменений и дополнений в техническое задание на последующих стадиях разработки программы или программного изделия выпускают дополнение к нему. Согласование и утверждение дополнения к техническому заданию проводят в том же порядке, который установлен для технического задания.

1.4. Техническое задание должно содержать следующие разделы:

- введение;
- наименование и область применения;
- основание для разработки;
- назначение разработки;
- технические требования к программе или программному изделию;
- технико-экономические показатели;
- стадии и этапы разработки;

- порядок контроля и приемки;
- приложения.

В зависимости от особенностей программы или программного изделия допускается уточнять содержание разделов, вводить новые разделы или объединять отдельные из них. При необходимости допускается в техническое задание включать приложения.

2. Содержание разделов

2.1 Введение должно включать краткую характеристику области применения программы или программного продукта, а также объекта (например, системы), в котором предполагается их использовать. Основное назначение введения - продемонстрировать актуальность данной разработки и показать, какое место эта разработка занимает в ряду подобных.

2.2 В разделе «Наименование и область применения» указывают наименование, краткую характеристику области применения программы или программного изделия и объекта, в котором используют программу или программное изделие.

2.3 В разделе «Основание для разработки» должны быть указаны:

- документ (документы), на основании которых ведется разработка. Таким документом может служить план, приказ, договор и т. п.;
- организация, утвердившая этот документ, и дата его утверждения;
- наименование и (или) условное обозначение темы разработки.

2.4 В разделе «Назначение разработки» должно быть указано функциональное и эксплуатационное назначение программы или программного изделия.

2.5 Раздел «Технические требования к программе или программному изделию» должен содержать следующие подразделы:

- требования к функциональным характеристикам;
- требования к надежности;
- условия эксплуатации;
- требования к составу и параметрам технических средств;
- требования к информационной и программной совместимости;
- требования к маркировке и упаковке;
- требования к транспортированию и хранению;
- специальные требования.

В подразделе «Требования к функциональным характеристикам» должны быть указаны требования к составу выполняемых функций, организации входных и выходных данных, временным характеристикам и т. п.

2.5.2 В подразделе «Требования к надежности» должны быть указаны требования к обеспечению надежного функционирования (обеспечение устойчивого функционирования, контроль входной и выходной информации, время восстановления после отказа и т. п.).

2.5.3 В подразделе «Условия эксплуатации» должны быть указаны условия эксплуатации (температура окружающего воздуха, относительная влажность и т. п. для выбранных типов носителей данных), при которых должны обеспечиваться заданные характеристики, а также вид обслуживания, необходимое количество и квалификация персонала.

2.5.4 В подразделе «Требования к составу и параметрам технических средств» указывают необходимый состав технических средств с указанием их технических характеристик.

2.5.5 В подразделе «Требования к информационной и программной совместимости» должны быть указаны требования к информационным структурам на входе и выходе и методам решения, исходным кодам, языкам программирования. При необходимости должна обеспечиваться защита информации и программ.

2.5.6 В подразделе «Требования к маркировке и упаковке» в общем случае указывают требования к маркировке программного изделия, варианты и способы упаковки.

2.5.7 В подразделе «Требования к транспортированию и хранению» должны быть указаны для

программного изделия условия транспортирования, места хранения, условия хранения, условия складирования, сроки хранения в различных условиях.

2.5.8 В разделе «Технико-экономические показатели» должны быть указаны: ориентировочная экономическая эффективность, предполагаемая годовая потребность, экономические преимущества разработки по сравнению с лучшими отечественными и зарубежными образцами или аналогами.

2.6 В разделе «Стадии и этапы разработки» устанавливают необходимые стадии разработки, этапы и содержание работ (перечень программных документов, которые должны быть разработаны, согласованы и утверждены), а также, как правило, сроки разработки и определяют исполнителей.

2.7 В разделе «Порядок контроля и приемки» должны быть указаны виды испытаний и общие требования к приемке работы.

2.8 В приложениях к техническому заданию при необходимости приводят:

- перечень научно-исследовательских и других работ, обосновывающих разработку;
- схемы алгоритмов, таблицы, описания, обоснования, расчеты и другие документы, которые могут быть использованы при разработке;
- другие источники разработки.

В случаях, если какие-либо требования, предусмотренные техническим заданием, заказчик не предъявляет, следует в соответствующем месте указать «Требования не предъявляются».

Задания для практического занятия:

1. Разработать техническое задание по варианту, указанного преподавателем (Приложение 2). Пример технического задания на разработку «Модуля автоматизированной системы оперативно-диспетчерского управления теплоснабжением корпусов Московского института» приведен в Приложении 3.

2. Оформить отчет

Вопросы для закрепления теоретического материала к практическому занятию:

1. Назовите этапы разработки программного обеспечения.
2. Что включает в себя постановка задачи и предпроектные исследования?
3. Перечислите функциональные и эксплуатационные требования к программному продукту.
4. Перечислите правила разработки технического задания.
5. Назовите основные разделы технического задания

Порядок выполнения отчета по практической работе

1.Разработать техническое задание на программный продукт (см. варианты заданий в приложении 2).

2. Оформить работу в соответствии с ГОСТ 19.106-78. При оформлении использовать MS Office.

Отчет по практической работе должен состоять из:

1. Постановки задачи.
2. Технического задания на программный продукт.

3. Сдать и защитить работу

Защита отчета по практической работе заключается в предъявлении преподавателю полученных результатов (на экране монитора и печатном виде), демонстрации полученных навыков и ответах на вопросы преподавателя

Практическая работа №3

Разработка документации. Стадия «Эскизный проект»

Цель работы состоит в приобретении навыков создания формальных моделей и на их основе определение спецификаций разрабатываемого программного обеспечения

Краткие теоретические и учебно-методические материалы по теме практической работы

Эскизный проект

Эскизный проект предусматривает разработку предварительных проектных решений по системе и ее частям.

Выполнение стадии эскизного проектирования не является строго обязательной. Если основные проектные решения определены ранее или достаточно очевидны для конкретной ИС и объекта автоматизации, то эта стадия может быть исключена из общей последовательности работ.

Содержание эскизного проекта задается в ТЗ на систему. Как правило, на этапе эскизного проектирования определяются:

- функции ИС;
- функции подсистем, их цели и ожидаемый эффект от внедрения;
- состав комплексов задач и отдельных задач;
- концепция информационной базы и ее укрупненная структура;
- функции системы управления базой данных;
- состав вычислительной системы и других технических средств;
- функции и параметры основных программных средств.

По результатам проделанной работы оформляется, согласовывается и утверждается документация в объеме, необходимом для описания полной совокупности принятых проектных решений и достаточном для дальнейшего выполнения работ по созданию системы.

На основе технического задания (и эскизного проекта) разрабатывается технический проект.

Разработка эскизного проекта программы. Этапы выполнения эскизного проекта.

. Эскизный проект	Разработка эскизного проекта	Предварительная разработка структуры входных и выходных данных.
		Уточнение методов решения задачи.
		Разработка общего описания алгоритма решения задачи
		Разработка технико-экономического обоснования.
	Утверждение эскизного проекта	Разработка пояснительной записки.
		Согласование и утверждение эскизного проекта.

Основная *задача эскизного проекта* – создать прообраз будущей автоматизированной системы. При разработке эскизного проекта разработчик определяет основные контуры будущей системы, а заказчик в свою очередь получает представление об основных чертах будущего объекта автоматизации и анализирует их возможную применимость в последующей работе.

При разработке эскизного проекта составляются:

- Ведомость эскизного проекта. Общая информация по проекту.
- Пояснительная записка к эскизному проекту. Вводная информация, позволяющая ее потребителю быстро освоить данные по конкретному проекту.

- Схема организационной структуры. Описание организационной структуры организации, которая будет использовать создаваемую автоматизированную систему в практической работе.
- Структурная схема комплекса технических средств. Техническая составляющая автоматизированной системы, включающая в себя набор серверов, рабочих станций, схему локальной вычислительной сети и структурированной кабельной системы.
- Схема функциональной структуры. Описание задач, которые будут использоваться в работе подсистем. Видение участков информационной системы и порядок и их взаимодействия.
- Схема автоматизации. Логический процесс создания автоматизированной системы от начала до конца.
- Согласно ГОСТ 34.201-89, дополнительно в эскизный проект по необходимости может быть включено техническое задание на разработку новых технических средств.

Эскизный проект чаще всего не разделяют, он выполняется в рамках общего (первоначального) этапа всего проекта. Перечень работ, составляющих эскизный проект, может варьироваться в зависимости от конкретного технического задания заказчика (его пожеланий) и сложности проектируемого проекта. Соответственно варьируется и цена этого этапа.

Эскизный проект не всегда создается под конкретного заказчика. Нередко разработчики с помощью эскизного проекта стремятся показать свой творческий потенциал и найти потенциальных заказчиков. Не случайно на различные конкурсы представляются именно эскизные проекты**.**

Разработка спецификаций

Разработка программного обеспечения начинается с анализа требований к нему. В результате анализа получают спецификации разрабатываемого программного обеспечения, строят общую модель его взаимодействия с пользователем или другими программами и конкретизируют его основные функции.

При структурном подходе к программированию на этапе анализа и определения спецификаций разрабатывают три типа моделей: модели функций, модели данных и модели потоков данных. Поскольку разные модели описывают проектируемое программное обеспечение с разных сторон, рекомендуется использовать сразу несколько моделей, разрабатываемых в виде диаграмм, и пояснять их текстовыми описаниями, словарями и т. п.

Структурный анализ предполагает использование следующих видов моделей:

- диаграмм потоков данных (DFD - Data Flow Diagrams), описывающих взаимодействие источников и потребителей информации через процессы, которые должны быть реализованы в системе;
- диаграмм «сущность-связь» (ERD Entity-Relationship Diagrams), описывающих базы данных разрабатываемой системы;
- диаграмм переходов состояний (STD - State Transition Diagrams), характеризующих поведение системы во времени;
- функциональных диаграмм (методика SADT);
- спецификаций процессов;
- словаря терминов.

Спецификации процессов

Спецификации процессов обычно представляют в виде краткого текстового описания, схем алгоритмов, псевдокодов, Flow-форм или диаграмм Насси - Шнейдермана (см. [1] разд. 3.5.1).

Словарь терминов

Словарь терминов представляет собой краткое описание основных понятий, используемых при составлении спецификации. Он должен включать определение основных понятий предметной области, описание структур элементов данных, их типом и форматов, а также всех сокращений и условных обозначении (см. [1] разд. 3.5.2).

Диаграммы переходов состояний

С помощью *диаграмм переходов состояний* можно моделировать последующее функционирование системы на основе ее предыдущего и текущего функционирования. Моделируемая система в любой заданный момент времени находится точно в одном из конечного множества состояний. С течением времени она может изменить свое состояние, при этом переходы между состояниями должны быть точно определены (см. [1] разд. 3.5.3).

Функциональные диаграммы

Функциональные диаграммы отражают взаимосвязи функций разрабатываемого программного обеспечения.

Они создаются на ранних этапах проектирования систем, для того чтобы помочь проектировщику выявить основные функции и составные части проектируемой системы и, по возможности, обнаружить и устранить существенные ошибки. Для создания функциональных диаграмм предлагается использовать методологию SADT (см. [1] разд. 3.5.4).

Диаграммы потоков данных

Для описания потоков информации в системе применяются *диаграммы потоков данных* (DFD – Data Flow Diagrams). DFD позволяет описать требуемое поведение системы в виде совокупности процессов, взаимодействующих посредством связывающих их потоков данных. DFD показывает, как каждый из процессов преобразует свои входные потоки данных в выходные потоки данных и как процессы взаимодействуют между собой (см. разд. [1] 3.5.5).

Диаграммы «сущность - связь»

Диаграмма сущность-связь - инструмент разработки моделей данных, обеспечивающий стандартный способ определения данных и отношений между ними. Она включает *сущности* и взаимосвязи, отражающие основные бизнес-правила предметной области. Такая диаграмма не слишком детализирована, в нее включаются основные *сущности* и *связи* между ними, которые удовлетворяют требованиям, предъявляемым к ИС (см. разд. [1] 3.5.6).

Задания для практического занятия:

1. На основе технического задания из практической работы №1 выполнить анализ функциональных и эксплуатационных требований к программному продукту.
2. Определить основные технические решения (выбор языка программирования, структура программного продукта, состав функций ПП, режимы функционирования) и занести результаты в документ, называемый «Эскизным проектом» (см. приложение 4).
3. Используя Приложение 5, выполнить следующее (по заданию преподавателя):
   - Определить диаграммы потоков данных для решаемой задачи.
   - Определить диаграммы «сущность-связь», если программный продукт содержит базу данных.
   - Определить функциональные диаграммы.
   - Определить диаграммы переходов состояний.
   - Определить спецификации процессов.
4. Добавить словарь терминов.
5. Оформить результаты, используя MS Office или MS Visio в виде эскизного проекта.
6. Сдать и защитить работу.

Вопросы для закрепления теоретического материала к практическому занятию:

1. Назовите этапы разработки программного обеспечения.
2. Что такое жизненный цикл программного обеспечения?
3. В чем заключается постановка задачи и предпроектные исследования?
4. Назовите функциональные и эксплуатационные требования к программному продукту.
5. Перечислите составляющие эскизного проекта.
6. Охарактеризуйте спецификации и модели.

Порядок выполнения отчета по практической работе

Отчет по практической работе должен состоять из:

1. Постановки задачи.
2. Документа «Эскизный проект», содержащего:

- выбор метода решения и языка программирования;
- спецификации процессов;
- все полученные диаграммы;
- словарь терминов.

Защита отчета по практической работе заключается в предъявлении преподавателю полученных результатов (на экране монитора), демонстрации полученных навыков и ответах на вопросы преподавателя.

Практическая работа№ 4

Разработка документации. Стадия «Технический проект».

Разработка логической модели данных

Цель работы состоит в приобретении навыков проектирования программного обеспечения

Краткие теоретические и учебно-методические материалы по теме практической работы

ПРОЕКТ ТЕХНИЧЕСКИЙ - образ намеченного к созданию объекта, представленный в виде его описания, схем, чертежей, расчетов, обоснований, числовых показателей.

Цель технического проекта - определение основных методов, используемых при создании информационной системы, и окончательное определение ее сметной стоимости.

Техническое проектирование подсистем осуществляется в соответствии с утвержденным техническим заданием.

Технический проект программной системы подробно описывает:

- выполняемые функции и варианты их использования;
- соответствующие им документы;
- структуры обрабатываемых баз данных;
- взаимосвязи данных;
- алгоритмы их обработки.

Технический проект должен включать данные об объемах и интенсивности потоков обрабатываемой информации, количестве пользователей программной системы, характеристиках оборудования и программного обеспечения, взаимодействующего с проектируемым программным продуктом.

При разработке технического проекта оформляются:

- ведомость технического проекта. Общая информация по проекту;
- пояснительная записка к техническому проекту. Вводная информация, позволяющая ее потребителю быстро освоить данные по конкретному проекту;
- описание систем классификации и кодирования;
- перечень входных данных (документов). Перечень информации, которая используется как входящий поток и служит источником накопления;
- перечень выходных данных (документов). Перечень информации, которая используется для анализа накопленных данных;

- описание используемого программного обеспечения. Перечень программного обеспечения и СУБД, которые планируется использовать для создания информационной системы;
- описание используемых технических средств. Перечень аппаратных средств, на которых планируется работа проектируемого программного продукта;
- проектная оценка надежности системы. Экспертная оценки надежности с выявлением наиболее благополучных участков программной системы и ее узких мест;
- ведомость оборудования и материалов. Перечень оборудования и материалов, которые потребуются в ходе реализации проекта.

Структурная схема

Структурной называют схему, отражающую состав и взаимодействие по управлению частями разрабатываемого программного обеспечения. Структурная схема определяется архитектурой разрабатываемого ПО (см. [1] разд. 4.1.1).

Функциональная схема

Функциональная схема - это схема взаимодействия компонентов программного обеспечения с описанием информационных потоков, состава данных в потоках и указанием используемых файлов и устройств (см. [1] разд. 4.1.2).

Разработка алгоритмов

Метод пошаговой детализации реализует нисходящий подход к программированию и предполагает пошаговую разработку алгоритма (см. [1] разд. 4.1.3).

Структурные карты

Методика структурных карт используется на этапе проектирования ПО для того, чтобы продемонстрировать, каким образом программный продукт выполняет системные требования. Структурные карты Константайна предназначены для описания отношений между модулями (см. [1] разд. 4.2).

Техника структурных карт Джексона основана на методе структурного программирования Джексона, который выявляет соответствие между структурой потоков данных и структурой программы. Основное внимание в методе сконцентрировано на соответствии входных и выходных потоков данных (см. [1] разд. 4.3).

Задания для практического занятия:

1. На основе технического задания из практической работы № 2 и спецификаций из практической работы № 3 разработать уточненные алгоритмы программ, составляющих заданный программный модуль. Использовать метод пошаговой детализации (см. [1] разд. 4.1.3).
2. На основе уточненных и доработанных алгоритмов разработать структурную схему программного продукта ([1] разд. 4.1.1).
3. Разработать функциональную схему программного продукта (разд. [1] 4.1.2).
4. Представить структурную схему в виде структурных карт Константайна (см. [1] разд. 4.2).
5. Представить структурную схему в виде структурных карт Джексона (см. [1] разд. 4.3).
6. Оформить результаты, используя MS Office или MS Visio в виде технического проекта.
7. Сдать и защитить работу.

Вопросы для закрепления теоретического материала к практическому занятию:

1. Назовите этапы разработки программного обеспечения.
2. В чем заключается проектирование программного обеспечения?
3. Перечислите составляющие технического проекта.
4. Охарактеризуйте структурный подход к программированию.
5. Из чего состоят структурная и функциональная схемы?

6. Охарактеризуйте метод пошаговой детализации при составлении алгоритмов программ.
7. Приведите понятие псевдокода.
8. В чем заключается методика Константайна?
9. В чем заключается методика Джексона?

Порядок выполнения отчета по практической работе

Отчет по практической работе должен состоять из:
1. Структурной схемы программного продукта.
2. Функциональной схемы.
3. Алгоритма программы.
4. Структурной карты Константайна.
5. Структурной карты Джексона.
6. Законченного технического проекта программного модуля.

Защита отчета по практической работе заключается в предъявлении преподавателю полученных результатов (на экране монитора и в печатном виде), демонстрации полученных навыков и ответах на вопросы преподавателя.

Практическая работа №5

Применение методов объектно-ориентированного проектирования

Учебная цель:

Цель работы состоит в приобретении навыков

Краткие теоретические и учебно-методические материалы по теме практической работы

Сущность объектно-ориентированного подхода к программированию заключается в том, что основные идеи объектно-ориентированного подхода опираются на следующие положения:

- программа представляет собой модель некоторого реального процесса, части реального мира;
- модель реального мира или его части может быть описана как совокупность взаимодействующих между собой объектов;
- объект описывается набором параметров, значения которых определяют состояние объекта, и набором операций (действий), которые может выполнять объект;
- взаимодействие между объектами осуществляется посылкой специальных сообщений от одного объекта к другому. Сообщение, полученное объектом, может потребовать выполнения определенных действий, например, изменения состояния объекта;
- объекты, описанные одним и тем же набором параметров и способные выполнять один и тот же набор действий представляют собой класс однотипных объектов.

С точки зрения языка программирования класс объектов можно рассматривать как тип данного, а отдельный объект - как данное этого типа. Определение программистом собственных классов объектов для конкретного набора задач должно позволить описывать отдельные задачи в терминах самого класса задач (при соответствующем выборе имен типов и имен объектов, их параметров и выполняемых действий).

Таким образом, объектно-ориентированный подход предполагает, что при разработке программы должны быть определены классы используемых в программе объектов и построены их описания, затем созданы экземпляры необходимых объектов и определено взаимодействие между ними.

Классы объектов часто удобно строить так, чтобы они образовывали иерархическую структуру. Например, класс «Студент», описывающий абстрактного студента, может служить основой для построения классов «Студент 1 курса», «Студент 2 курса» и т.д., которые обладают всеми свойствами студента вообще и некоторыми дополнительными свойствами, характеризующими студента конкретного курса. При разработке интерфейса с пользователем программы могут использовать объекты общего класса «Окно» и объекты классов специальных окон, например, окон информационных сообщений, окон ввода данных и т.п. В таких иерархических структурах один класс может рассматриваться как базовый для других, производных от него классов. Объект производного класса обладает всеми свойствами базового класса и некоторыми собственными свойствами, он может реагировать на те же типы сообщений от других объектов, что и объект базового класса и на сообщения, имеющие смысл только для производного класса. Обычно говорят, что объект производного класса наследует все свойства своего базового класса.

Некоторые параметры объекта могут быть локализованы внутри объекта и недоступны для прямого воздействия извне объекта. Например, во время движения объекта-автомобиля объект-водитель может воздействовать только на ограниченный набор органов управления (рулевое колесо, педали газа, сцепления и тормоза, рычаг переключения передач) и ему недоступен целый ряд параметров, характеризующих состояние двигателя и автомобиля в целом.

Очевидно, для того, чтобы продуктивно применять объектный подход для разработки программ, необходимы языки программирования, поддерживающие этот подход, т.е. позволяющие строить описание классов объектов, образовывать данные объектных типов, выполнять операции над объектами. Одним из первых таких языков стал язык SmallTalk в котором все данные являются объектами некоторых классов, а общая система классов строится как иерархическая структура на основе предопределенных базовых классов.

Опыт программирования показывает, что любой методический подход в технологии программирования не должен применяться слепо с игнорированием других подходов. Это относится и к объектно-ориентированному подходу. Существует ряд типовых проблем, для которых его полезность наиболее очевидна, к таким проблемам относятся, в частности, задачи имитационного моделирования, программирование диалогов с пользователем. Существуют и задачи, в которых применение объектного подхода ни к чему, кроме излишних затрат труда, не приведет. В связи с этим наибольшее распространение получили объектно-ориентированные языки программирования, позволяющие сочетать объектный подход с другими методологиями. В некоторых языках и системах программирования применение объектного подхода ограничивается средствами интерфейса с пользователем (например, Visual FoxPro ранних версий).

Наиболее используемыми в настоящее время объектно-ориентированными языками являются Паскаль с объектами и Си++, причем наиболее развитые средства для работы с объектами содержатся в Си++.

Объектно-базирующееся программирование - это методология разработки программ, основанная на использовании совокупности объектов, каждый из которых является реализацией определенного класса. Программный код и данные структурируются так, чтобы имитировалось поведение фактически существующих объектов. Содержимое объекта защищено от внешнего мира посредством инкапсуляции. Благодаря наследованию уже запрограммированные функциональные возможности можно использовать и для других объектов. Объекты являются программным представлением физических и/или логических сущностей реального мира. Они необходимы для моделирования поведения физических или логических объектов, которые они представляют. Для изменения поведения и состояния элементов управления используются их свойства, методы, поля и события. Классы задают структуру объектов. При программировании создаются объекты - представители классов. С другой стороны, классы составляют группы одноименных объектов. Внутренняя структура класса в Visual Basic передается объекту с

использованием модуля класса. С использованием команды Project ⟶Add Class Module модуль класса можно добавить в проект. После добавления модуля класса выводится окно кода, в котором можно реализовать компоненты (свойства, поля, методы, события) класса.

Пример использования методики объектно–ориентированного программирования

Задание. Создать в предметной области «Автомобили» класс с требуемой функциональностью (использовать компоненты класса: методы, поля и т.д.). Создать объект - экземпляр класса. Создать пример использования объектом компонентов класса.

Реализация задания

Приводится проект, дающий справку желающим приобрести автомобиль. Создан класс Class1, содержащий компоненты, определяющие название фирмы-изготовителя, модель автомобиля, его стоимость, изображение автомобиля и следующие технические характеристики:

- тип двигателя (бензин/дизель),
- число цилиндров/рабочий объём,
- система питания (карбюратор/впрыскивание),
- мощность (л.с),
- максимальная скорость (км/час),
- разгон 0 - 100 (км/час)/сек,
- привод (передний/задний/4х4).

Далее создаётся экземпляр класса: Dim av As New Class1, использующий компоненты класса.

Пользователю предлагается решить вопрос о необходимости покупки, выбрать фирму-изготовителя, ответить на вопрос о выводе изображения покупаемого автомобиля, либо его технических характеристик, либо обеих категорий одновременно (используются процедуры Property Get и Property Let, созданные в классе Class1), после чего программа адекватно реагирует: либо выводятся вышеперечисленные данные, либо выводится некоторое сообщение.

Для реализации проекта нужно выполнить следующую последовательность действий:

1. добавить в стандартный проект модуль класса (Project ⟶Add Class Module ⟶Class Module ⟶ Открыть),
2. создать:
   - методы класса. Четыре метода создаются в процедурах: Public Function Met1(), Public Function Met2(), Public Function Met3(), Public Function Met4() (Tools ⟶Add Procedure ⟶ввести имя { Met1, Met2, Met3, Met4}⟶ выбрать Function ⟶выбрать Public ⟶ОК),
   - свойства класса. Свойства задаются с использованием процедур Property Get и Property Let (Tools ⟶ Add Procedure ?ввести имя (здесь - varian) ⟶выбрать Property ⟶выбрать Public ⟶ОК),
   - поля класса - avto, firma, model, stoim, pict, var, см. ниже.
3. создать на форме:
   - два элемента управления ComboBox с именами Combo1 и Combo2,
   - два элемента управления CommandButton с именами Command1 и Command2; значению свойства Caption объекта Command1 присвоить значение "OK", Command2 - "Exit",
   - элементы управления Label1 - Label4, значениям свойств Caption присвоить: Label1 - "Хотите ли Вы купить машину?", Label2 - "Выберите фирму-изготовитель", Label3 - "Хотите ли Вы увидеть изображение выбранного автомобиля или его технические характеристики ?", Label4- "", свойству Visible объекта Label4 присвоить False,
   - массив элементов управления OptionButton (присвоить значения свойствам - Option1(0).Caption= "да", Option1(1).Caption= "нет"),
   - массивы элементов управления PictureBox: Picture1(0) - Picture1(12) и Picture2(0) - Picture(12). Свойству Visible всех элементов управления присвоить значение False. Свойству Picture каждого элемента управления присвоить значение изображения соответствующего автомобиля и списка технических характеристик (эти списки создаются в приложении Excel, далее таблицы передаются в приложение Paint и сохраняются как рисунки).
4. ввести код в область класса (см. ниже "область проекта Class1"),
5. ввести код, данный ниже, в области:
   - General Declarations формы,
   - Combo1, событие Click,
   - Combo2, событие Click,
   - Command1, событие Click,
   - Command, событие Click,
   - Form, событие Load,

- Form, событие Unload,
6. стартовать проект, получить справку о предполагаемой покупке.

```
//////////////////////////область проекта Class1////////////////////////////

Public avto As Boolean

Public firma As String

Public model As String

Public stoim As String

Public pict As String

Dim var As String

Private Sub Class_Initialize() ' инициализация полей класса

avto = False: firma = "": model = "": stoim = "": var = ""

End Sub

Public Function Met1()

' Если пользователь нажал кнопку (OptionButton) - Да, то выполнить процедуры

' Met2, Met3, Met4, результатом выполнения которых является вывод данных:

' марка, стоимость, изображение и технические характеристики, иначе

' Met1 = False и выводится сообщение "Приносим свои извинения, мы даем

' информацию для желающих купить автомобиль"

If avto = True Then

model = Met2()

stoim = Met3()

pict = Met4() ' поле pict определяет номера элементов массивов

' PictureBox, см. Met4

Met1 = True

Else

Met1 = False

End If

End Function

' после щелчка на кнопках Да/Нет (два переключателя OptionButton) и выбора

' фирмы из списка ComboBox с именем Combo1 определить марку автомобиля

Public Function Met2()

Select Case firma
```

```
Case "AUDI": Met2 = "A6"
Case "CITROEN": Met2 = "C5"
Case "FORD": Met2 = "Focus"
Case "HONDA": Met2 = "Accord"
Case "HYUNDAI": Met2 = "Elanta"
Case "JEEP": Met2 = "Grand Cherokee LTD"
Case "LAND ROVER": Met2 = "Land Rover Discovery"
Case "LEXSUS": Met2 = "RX330"
Case "MITSUBISHI": Met2 = "Pajero III"
Case "NISSAN": Met2 = "Primera(1.8)"
Case "PEUGEOT": Met2 = "307 XR"
Case "PORSCHE": Met2 = "Cayenne Turbo"
Case "RENAULT": Met2 = "Laguna II"
End Select
End Function
' определить стоимость автомобиля в долларах США
Public Function Met3()
Select Case firma
Case "AUDI": Met3 = "41500"
Case "CITROEN": Met3 = "20100"
Case "FORD": Met3 = "12430"
Case "HONDA": Met3 = "33900"
Case "HYUNDAI": Met3 = "13790"
Case "JEEP": Met3 = "41690"
Case "LAND ROVER": Met3 = "40850"
Case "LEXSUS": Met3 = "65500"
Case "MITSUBISHI": Met3 = "56640"
Case "NISSAN": Met3 = "25100"
Case "PEUGEOT": Met3 = "13808"
Case "PORSCHE": Met3 = "140500"
Case "RENAULT": Met3 = "22900"
```

```
End Select
End Function
Public Function Met4()
' при выборе данных из списка ComboBox с именем Combo2
' (после щелчка на кнопке "ОК" ) определяется номер элемента массива
' PictureBox, соответствующий выбранной фирме-изготовителю и
' на экран позднее выводится соответствующая фотография
' и/или технические характеристики автомобиля
Select Case firma
Case "AUDI": Met4 = "0"
Case "CITROEN": Met4 = "1"
Case "FORD": Met4 = "2"
Case "HONDA": Met4 = "3"
Case "HYUNDAI": Met4 = "4"
Case "JEEP": Met4 = "5"
Case "LAND ROVER": Met4 = "6"
Case "LEXSUS": Met4 = "7"
Case "MITSUBISHI": Met4 = "8"
Case "NISSAN": Met4 = "9"
Case "PEUGEOT": Met4 = "10"
Case "PORSCHE": Met4 = "11"
Case "RENAULT": Met4 = "12"
End Select
End Function
' процедура Property Let используется для задания значения свойства,
' Property Get - для считывания значения свойства
Public Property Get varian() As String
Select Case var
Case Is = 0: varian = "pict"
Case Is = 1: varian = "texn"
Case Is = 2: varian = "all"
```

```
End Select
End Property
Public Property Let varian(ByVal vNewValue As String)
Select Case vNewValue
Case "изображение": var = 0
Case "технические параметры": var = 1
Case Else: var = 2
End Select
End Property
////////////////////////////////область проекта Form1//////////////////////////////
Dim av As Class1 ' av - экземпляр класса
Dim v As String
Dim i As Integer, j As Integer
Private Sub Combo1_Click()
' сделать невидимыми элементы управления Label и Picture
' (формирующие фотографии, технические характеристики, фирму,
' марку и стоимость), для того, чтобы впоследствии на форму
' выводились только те из них, которые определяет своими
' действиями покупатель
Label5.Visible = False
For i = 0 To 12
Picture1(i).Visible = False
Picture2(i).Visible = False
Next
Dim ot As String ' переменная для хранения сообщения программы
av.firma = Combo1.Text ' значение поля firma объекта av взять из
' списка ComboBox с именем Combo1
av.avto = Option1(0).Value ' значение поля avto объекта av взять
' из поля массива OptionButton
If av.Met1 = True Then
ot = " " & CStr(av.firma) & vbCrLf: ot = ot & " " & vbCrLf
```

```
ot = ot & " модель " & CStr(av.model) & vbCrLf: ot = ot & " " & vbCrLf
ot = ot & " цена в $ " & CStr(av.stoim) & vbCrLf: ot = ot & " " & vbCrLf
ot = ot & "Для получения более полной информации обращайтесь_
по телефону 7077888"
MsgBox Title:="Мы можем предложить", Prompt:=ot
Else
Label5.Visible = False
Picture1(Val(av.pict)).Visible = False ' аргумент Picture1: (av.pict)
' определяет индекс элемента массива PictureBox
ot = "Приносим свои извинения, мы даем информацию для желающих_
купить автомобиль"
MsgBox Title:="Автосалон START", Prompt:=ot
End If
End Sub
Private Sub Combo2_Click()
av.varian = Combo2.Text ' см. процедуру Property Let. Присваиваем
' свойству varian значение выбранные из списка ComboBox с именем Combo2
End Sub
Private Sub Command1_Click()
Label5.Visible = False
Label5.Caption = ""
For i = 0 To 12
Picture1(i).Visible = False
Picture2(i).Visible = False
Next
v = av.varian ' см. процедуру Property Get. Переменной v присваиваем
' значение свойства varian объекта av
av.avto = Option1(0).Value
If av.Met1 = True Then
Select Case v
Case "pict"
```

```
Picture1(Val(av.pict)).Visible = True
Case "texn"
Picture2(Val(av.pict)).Visible = True ' технические характеристики
' хранятся как изображения в соответствующих элементах
' массива PictureBox2
Case "all"
Picture1(Val(av.pict)).Visible = True
Picture2(Val(av.pict)).Visible = True
Label5.Visible = True
Label5.Caption = CStr(av.firma) & " " & CStr(av.model) & _
vbCrLf & "цена в $ " & CStr(av.stoim)
End Select
Else
Picture1(Val(av.pict)).Visible = False
Picture2(Val(av.pict)).Visible = False
MsgBox Title:="Автосалон START", Prompt:="Приносим свои_
извинения, мы даем информацию для желающих купить автомобиль"
End If
End Sub
Private Sub Command2_Click()
MsgBox Title:="Автосалон START", Prompt: = "Мы всегда рады помочь!_
Будем рады новой встрече!"
End ' выход из программы после сообщения MsgBox.
End Sub
Private Sub Form_Load()
Set av = New Class1 ' описание объектной переменной дано выше
' заполнение списка ComboBox с именем Combo1 названиями фирм
Combo1.AddItem "AUDI"
Combo1.AddItem "CITROEN"
Combo1.AddItem "FORD"
Combo1.AddItem "HONDA"
```

```
Combo1.AddItem "HYUNDAI"

Combo1.AddItem "JEEP"

Combo1.AddItem "LAND ROVER"

Combo1.AddItem "LEXSUS"

Combo1.AddItem "MITSUBISHI"

Combo1.AddItem "NISSAN"

Combo1.AddItem "PEUGEOT"

Combo1.AddItem "PORSCHE"

' заполнение списка ComboBox с именем Combo2 предложениями для

' выбора данных в процедурах Property Get и Property Let

Combo2.AddItem "изображение"

Combo2.AddItem "технические параметры"

Combo2.AddItem "все данные"

End Sub

Private Sub Form_Unload(Cancel As Integer)

Set av = Nothing ' удалить объект из памяти

End Sub
```

Инструкция пользователя

После старта проекта при отрицательном ответе на вопрос «Хотите ли Вы купить машину?» выводится сообщение: «Приносим свои извинения, мы даем информацию для желающих купить автомобиль».

При положительном ответе на вопрос и выборе фирмы-изготовителя из списка на экран выводится сообщение о фирме, марке и стоимости автомобиля.

Далее покупатель может просмотреть или внешний вид, или технические характеристики, или одновременно обе категории, выбрав соответствующую строку во втором списке и сделав щелчок на кнопке ОК (используются процедуры Property Get и Property Let), где дан результат работы программы при выборе строки «все данные».

При щелчке на кнопке Exit выводится сообщение: «Мы всегда рады помочь! Будем рады новой встрече!» и проводится выход из программы.

Заключение (выводы)

Созданный программный продукт позволяет клиенту получить справочные данные при покупке автомобиля. Представленная программа является лишь небольшим примером использования классов, в реальности же сфера применения свойств объектно-базирующегося программирования гораздо шире.

Задания для выполнения:

1. Для предметной области (заданной преподавателем к практической работе №2) выполнить объектно-ориентированное проектирование программного продукта.

2. Оформить отчет.

Отчет по практической работе должен быть оформлен на основании требований и состоять из следующих структурных элементов:

- Анализа предметной области
- Определения функций предметной области
- Схемы документопотока
- Выделенных сущностей, атрибутов и установленных связей
- Концептуальной модели
- Описания выходных и входных данных

3. Ответить на контрольные вопрсы

Вопросы для закрепления теоретического материала к практическому занятию:

1. В чем заключается сущность объектно-ориентированного подхода при разработке программного продукта?
2. На что направлен объектно-ориентированный анализ?
3. Перечислите основные достоинства объектно-ориентированной методологии по сравнению со структурными методами.
4. Перечислите принципы объектного подхода. Дайте им краткие характеристики
5. Назовите основные методики объектно-ориентированного анализа.
6. Какие понятия, необходимые для проектирования системы включает концептуальная модель?

Порядок выполнения отчета по практической работе

Защита отчета по практической работе заключается в предъявлении преподавателю полученных результатов, демонстрации полученных навыков и ответах на вопросы преподавателя.

Практическая работа № 6

Тестирование программного продукта методом «белого ящика»

Цель работы состоит в приобретении навыков тестирования логики программы, формализованного описания результатов тестирования и применению стандартов по составлению схем программ

Краткие теоретические и учебно-методические материалы по теме практической работы

Тестирование программного обеспечения включает в себя целый комплекс действий, аналогичных последовательности процессов разработки программного обеспечения. В него входят:

- постановка задачи для теста;
- проектирование теста;
- написание тестов;
- тестирование тестов;
- выполнение тестов;
- изучение результатов тестирования.

Наиболее важным является проектирование тестов. Существуют разные подходы к проектированию тестов.

Первый состоит в том, что тесты проектируются на основе внешних спецификаций программ и модулей либо спецификаций сопряжения модуля с другими модулями, программа при этом рассматривается как «черный ящик». Смысл теста заключается в том, чтобы проверить, соответствует ли программа внешним спецификациям. При этом содержание модуля не имеет значения. Такой подход получил название - **стратегия «черного ящика».**

Второй подход - **стратегия «белого ящика»,** основан на анализе логики программы. При таком подходе тестирование заключается в проверке каждого пути, каждой ветви алгоритма. При

этом внешняя спецификация во внимание не принимается.

Ни один из этих подходов не является оптимальным. Реализация тестирования методом **«черного ящика»** сводится к проверке всех возможных комбинаций входных данных. Невозможно протестировать программу, подавая на вход бесконечное множество значений, поэтому ограничиваются определенным набором данных. При этом исходят из максимальной отдачи теста по сравнению с затратами на его создание. Она измеряется вероятностью того, что тест выявит ошибки, если они имеются и программе. Затраты измеряются временем и стоимостью подготовки, выполнения и проверки результатов теста.

Тестирование методом **«белого ящика»** также не даст 100%-ной гарантии того, что модуль не содержит ошибок. Даже если предположить, что выполнены тесты для всех ветвей алгоритма, нельзя с полной уверенностью утверждать, что программа соответствует ее спецификациям. Например, если требовалось написать программу для вычисления кубического корня, а программа фактически вычисляет корень квадратный, то реализации будет совершенно неправильной, даже если проверить все пути. Вторая проблема — отсутствующие пути. Если программа реализует спецификации не полностью (например, отсутствует такая специализированная функция, как проверка на отрицательное значение входных данных программы вычисления квадратного корня), никакое тестирование существующих путей не выявит такой ошибки. И наконец, проблема зависимости результатом тестирования от входных данных. Одни данные будут давать правильные результаты, а другие нет. Например, если для определения равенства трех чисел программируется выражение вида:

IF (A + B+ C)/3 = A,

то оно будет верным не для всех значений *A, B* и *C* (ошибка возникает в том случае, когда из двух значений *B* или *C* одно больше, а другое на столько же меньше *A).* Если концентрировать внимание только на тестировании путей, нет гарантии, что эта ошибка будет выявлена.

Таким образом, полное тестирование программы невозможно, т. е. никакое тестирование не гарантирует полное отсутствие ошибок в программе. Поэтому необходимо проектировать тесты таким образом, чтобы увеличить вероятность обнаружения ошибки в программе.

Стратегия «белого ящика»

Существуют следующие методы тестирования по принципу «белого ящика»:

- покрытие операторов;
- покрытие решений;
- покрытие условий;
- покрытие решений/условий;
- комбинаторное покрытие условий.

Метод покрытия операторов

Целью этого метода тестирования является выполнение каждого оператора программы хотя бы один раз.

Пример.

Если для тестирования задать значения переменных A = 2, B=0, X= 3, будет реализован путь *ace,* т. е. каждый оператор программы выполнится один раз (рис.1, *а).* Но если внести в алгоритм ошибки - заменить в первом условии ***and*** на ***or,*** а во втором *X*> 1 на *X*< 1 (рис.1, *б),* ни одна ошибка не будет обнаружена (табл.1). Кроме того, путь *abd* вообще не будет охвачен тестом, и если в нем есть ошибка, она также не будет обнаружена. В табл.1 ожидаемый результат определяется по блок-схеме на рис.1, *а,* а фактический - по рис.1, *б.*

Как видно из этой таблицы, ни одна из внесенных в алгоритм ошибок не будет обнаружена.

Таблица 1 - Результат тестирования методом покрытия операторов

Тест	Ожидаемый результат	Фактический	Результат тести-рования
A = 2, B = 0, X=3	X=2,5	X=2,5	Неуспешно

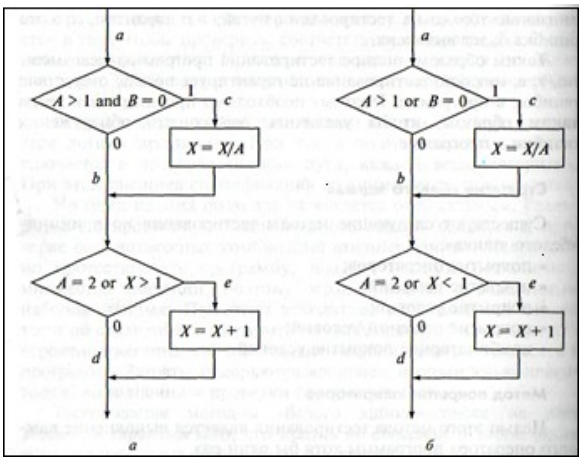


Рис. 1. Пример алгоритма программы: *а* - правильный; *б* - с ошибкой

Метод покрытия решений (покрытия переходов)

Согласно методу покрытия решений каждое направление перехода должно быть реализовано, по крайней мере, один раз. Этот метод включает в себя критерий покрытия операторов, так как при выполнении всех направлений переходов выполнятся все операторы, находящиеся на этих направлениях.

Для программы, приведенной на рис.1, покрытие решений может быть выполнено двумя тестами, покрывающими пути *{ace, abd},* либо *{acd, abe}.* Для этого выберем следующие исходные данные: *{A = 3, B*=0, *X*= 3} - в первом случае и {*A=2, B =1, X= 1}* - во втором. Однако путь, где *X* не меняется, будет проверен с вероятностью 50%: если во втором условии вместо условия *X*> 1 записано *X*< 1, то ошибка не будет обнаружена двумя тестами.

Результаты тестирования приведены в табл.2.

Таблица 2 - Результат тестирования методом покрытия решений

Тест	Ожидаемый результат	Фактический результат	Результат тестирования
A = 3, *B* = 0, *X* = 3	*X* = 1	*X* = 1	Неуспешно
A = 2, *B* = 1, *X* = 1	*X* = 2	*X* = 1,5	Успешно

Метод покрытия условий

Этот метод может дать лучшие результаты по сравнению с предыдущими. В соответствии с методом покрытия условий записывается число тестов, достаточное для того, чтобы все возможные результаты каждого условия в решении выполнялись, по крайней мере, один раз.

В рассматриваемом примере имеем четыре условия: *{A > 1, B = 0}, {A* = 2, *X*> 1}. Следовательно, требуется достаточное число тестов, такое, чтобы реализовать ситуации, где *A> 1, A≤ 1, B=0* и B≠ 0 в точке *a* и *A* = 2, *A* ≠ 2, *X*> 1 и *X*≤ 1 в точке *b.* Тесты, удовлетворяющие

критерию покрытия условий (табл.3), и соответствующие им пути:

а) *A = 2, B=0, X=4 ace;*

б)A = 1, *B= 1, X=0 abd.*

Таблица 3 - Результаты тестирования методом покрытия условий

Тест	Ожидаемый результат	Фактический результат	Результат тестирования
$A = 2,\ B = 0,\ X = 4$	$X = 3$	$X = 3$	Неуспешно
$A = 1,\ B = 1,\ X = 0$	$X = 0$	$X = 1$	Успешно

Метод покрытия решений (покрытия переходов)

Критерий покрытия решений/условий требует такого достаточного набора тестов, чтобы все возможные результаты каждого условия выполнялись по крайней мере один раз, все результаты каждого решения выполнялись по крайней мере один раз и, кроме того, каждой точке входа передавалось управление по крайней мере один раз.

Недостатки метода:

- не всегда можно проверить все условия;
- невозможно проверить условия, которые скрыты другими условиями;
- недостаточная чувствительность к ошибкам в логических выражениях.

Так, в рассматриваемом примере два теста метода покрытия условий

а) $A = 2,\ B = 0,\ X = 4$ *ace;*

б) $A = 1,\ B = 1,\ X = 0$ *abd*

отвечают и критерию покрытия решений/условий. Это является следствием того, что одни условия приведенных решений скрывают другие условия в этих решениях. Так, если условие $A > 1$ будет ложным, транслятор может не проверять условия $B = 0$, поскольку при любом результате условия $B=0$ результат решения $((A > 1)\&(B=0))$ примет значение *ложь.* То есть в варианте на рис.1 не все результаты всех условий выполнятся в процессе тестирования.

Рассмотрим реализацию того же примера на рис.2. Наиболее полное покрытие тестами в этом случае осуществляется

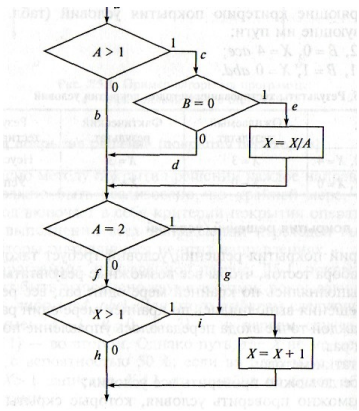


Рис.2. Пример алгоритма программы

так, чтобы выполнялись все возможные результаты каждого простого решения. Для этого нужно покрыть пути *aceg* (тест $A = 2,\ B=0,\ X= 4$), *acdfh* (тест $A = 3,\ B = 1,\ X= 0$), *abfh* (тест $A = 0, B=0,\ X= 0$), *abfi* (тест $A = 0,\ B= 0,\ X= 2$).

Протестировав алгоритм на рис.2, нетрудно убедиться в том, что критерии покрытия условий и критерии покрытия решений/условий недостаточно чувствительны к ошибкам в логических выражениях.

Метод комбинаторного покрытия условий

Критерий комбинаторного покрытия условий удовлетворяет также и критериям покрытия

решений, покрытия условий и покрытия решений/условий.

Этот метод требует создания такого числа тестов, чтобы все возможные комбинации результатов условия в каждом решении выполнялись по крайней мере один раз. По этому критерию в рассматриваемом примере должны быть покрыты тестами следующие восемь комбинаций:

1. A>1, B=0
2. A > 1, B≠0
3. A ≤1, B=0
4. A ≤ 1, B≠0
5. A = 2, X>1
6. A = 2, X≤1
7. A≠2, X>1
8. A ≠2,X≤1

Для того чтобы протестировать эти комбинации, необязательно использовать все 8 тестов. Фактически они могут быть покрыты четырьмя тестами (табл.4):

- *A* = 2, B=0, *X*= 4 {покрывает 1, 5};
- *A* = 2, B = 1, *X*= 1 {покрывает 2, 6};
- *A* = 0,5, B= 0, *X*= *2* {покрывает 3, 7};
- *A* = 1, B = 0, *X*= 1 {покрывает 4, 8}.

Таблица 4 - Результаты тестирования методом комбинаторного покрытия условий

Тест	Ожидаемый результат	Фактический результат	Результат тестирования
$A=2,\ B=0,\ X=4$	$X=3$	$X=3$	Неуспешно
$A=2,\ B=1,\ X=1$	$X=2$	$X=1{,}5$	Успешно
$A=0{,}5,\ B=0,\ X=2$	$X=3$	$X=4$	Успешно
$A=1,\ B=0,\ X=1$	$X=1$	$X=1$	Неуспешно

Задания для практического занятия:

1. Выполнить упражнения.

2.Оформить отчет по практической работе.

3. Сдать и защитить работу.

Упражнения

Задание 1.

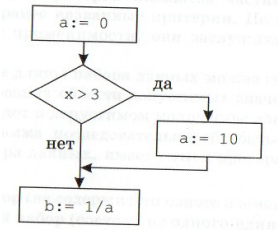
Первый из критериев белого ящика - критерий покрытия операторов. Он требует подобрать такой набор тестов, чтобы каждый оператор в программе был выполнен хотя бы один раз. В качестве примера рассмотрим следующий фрагмент Паскаль-программы:

Пример 1

```
a:= 0;
if x>3 then a:= 10;
b:= 1/a;
```

Для того чтобы удовлетворить критерию покрытия операторов, достаточно одного выполнения. Такого, чтобы *x* был больше 3. Очевидно, что ошибка в программе этим тестом обнаружена, не будет. Она проявится как раз в том случае, когда *x≤3.* Но такого теста критерий покрытия операторов от нас не требует.

Итак, мы имеем программу, оттестированную с точки зрения критерия покрытия операторов и при этом содержащую ошибку. Попробуем разобраться, в чем дело. Для наглядности перейдем с Паскаля на язык блок-схем.



Теперь причина видна сразу. Следуя критерию покрытия операторов, мы проверили только положительную ветвь раз вилки, но не затронули отрицательную. Сокращенная форма условного оператора в Паскале этому весьма способствует.

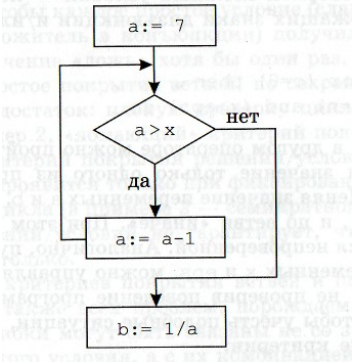
Чтобы избавиться от указанного недостатка, введем второй критерий белого ящика - **критерий покрытия ветвей** (иначе его называют критерием покрытия решений). Он требует подобрать такой набор тестов, чтобы каждая ветвь в программе была выполнена хотя бы один раз. Тестирование с точки зрения этого критерия обнаружит ошибку в предыдущем примере.

Рассмотрим другой пример. На Паскале он будет выглядеть так:

Пример 2

```
a:= 7;
while a>x do a:= a-1;
b:= 1/a;
```

Мы уже знаем, что паскалевская запись может служить провокатором ошибок. Поэтому сразу составим блок-схему:



Для того чтобы удовлетворить критерию покрытия ветвей, в данном случае достаточно одного теста. Например, такого, чтобы x был равен 6 или 5. Все ветви программы будут пройдены (при x=5 одна из ветвей - тело цикла - даже 2 раза). Но ошибка в программе обнаружена так и не будет! Она проявится в одном-единственном случае, когда $x = 0$. Но такого теста от нас критерий покрытия ветвей не потребовал.

Итак, мы имеем программу, оттестированную с точки зрения критерия покрытия ветвей и при этом содержащую ошибку. Причина в том, что некоторые ветви в программе могут быть пройдены несколько раз, и результат выполнения зависит от количества проходов. Для того чтобы учесть этот факт, введем третий критерий белого ящика - критерий **покрытия путей.** Он требует подобрать такой набор тестов, чтобы каждый путь в программе был выполнен хотя бы один раз. Тестирование с точки зрения этого критерия обнаружило бы ошибку в примере 2. Но из этого же примера виден принципиальный недостаток данного критерия. Сколько всего путей возможно в примере 2? Бесконечно много! Проверить их все невозможно. Значит, как только в программе появляются циклы с пред- или постусловием или цикл со счетчиком, но с вычисляемыми границами, количество путей в программе становится потенциально бесконечным, и критерий покрытия путей становится неприменимым. Необходим какой-то компромиссный критерий. Более жесткий, чем покрытие ветвей, но менее жесткий, чем покрытие путей. О нем мы поговорим далее.

Кроме проблем с проверкой циклов существенные проблемы связаны с проверкой сложных условий - логических выражений, содержащих знаки дизъюнкции и/или конъюнкции. Например:

```
if (a<b) or (c=0) then …
while (i<=n) and (x>eps) do …
```

И в том, и в другом операторе можно пройти по обеим ветвям, изменяя значение только одного из простых условий. Пусть $c \neq 0$. Меняя значение переменных a и b, можно пройти и по ветви «то», и по ветви «иначе». При этом ситуация, когда $c = 0$, останется непроверенной. Аналогично, пусть $i \leq n$. Меняя значения переменных x и eps, можно управлять выполнением цикла *while* не проверив поведение программы при $i > n$.

Для того чтобы учесть подобные ситуации, были предложены следующие критерии:

- критерий покрытия условий;
- критерий покрытия решений/условий;
- критерий комбинаторного покрытия условий.

Критерий **покрытия условий** требует подобрать такой набор тестов, чтобы каждое простое условие (слагаемое в дизъюнкции и сомножитель в конъюнкции) получило и значение «истина», и значение «ложь» хотя бы один раз.

Критерий пытается «в лоб» исправить вышеуказанный недостаток в тестировании сложных условий. Однако сам оказывается весьма слаб. Дело в том, что выполнение критерия покрытия условий не гарантирует покрытие ветвей. Пусть сложное условие представляет собой дизъюнкцию двух слагаемых. Например,

```
if (a<b) or (c=0) then d:= 1 else d:= 1/c;
```

При первом выполнении первое слагаемое истинно, второе ложно, вся дизъюнкция в целом истинна. При втором выполнении первое слагаемое ложно, второе истинно, вся дизъюнкция в целом истинна. Критерий покрытия условий выполнен, критерий покрытия ветвей – нет. Ошибка в программе не обнаружена.

Аналогичная ситуация возможна для конъюнкции. Например,

```
if (a<b) and (c=0) then d:= 1/c else d:= 1;
```

Чтобы исправить этот недостаток, критерии покрытия ветвей (решений) и условий объединяют в единый критерий **покрытия решений/условий.** Он требует подобрать такой набор тестов, чтобы каждая ветвь в программе была пройдена хотя бы один раз и чтобы каждое простое условие (слагаемое в дизъюнкции и сомножитель в конъюнкции) получило и значение «истина», и значение «ложь» хотя бы один раз. Критерий надежнее, чем простое покрытие ветвей, но сохраняет его принципиальный недостаток: плохую проверку циклов. Приведенный выше пример 2, «ломающий» критерий покрытия ветвей, «сломает» и критерий покрытия решений/условий. Ошибка в данном случае проявится только при фиксированном количестве повторений цикла (в примере 2 - семикратном), а критерий покрытия решений/условий не гарантирует, что повторений будет именно столько.

Совмещение критериев покрытия ветвей и покрытия условий не решает также всех проблем, порождаемых сложными условиями. Ошибки могут быть связаны не со значением того или иного простого условия, а с их комбинацией. Например, в фрагменте:

Пример 3

```
if (a=0) or (b=0) or (c=0)
   then d:= 1/(a+b)
   else d:= 1;
```

ошибка будет выявлена только при одновременном равенстве нулю двух переменных: a и b.

Для решения этой проблемы был предложен критерий **комбинаторного покрытия условий,** который требует подобрать такой набор тестов, чтобы хотя бы один раз выполнялась любая комбинация простых условий. Критерий значительно более надежен, чем покрытие

решений/условий, но обладает двумя существенными недостатками. Во-первых, он может потребовать очень большого числа тестов. Количество тестов, необходимых для проверки комбинации *n* простых условий, равно 2". Комбинация двух условий потребует четырех тестов, трех условий - восьми, четырех условий - шестнадцати тестов и т. д. Во-вторых, даже комбинаторное покрытие условий не гарантирует надежную проверку циклов. Тот же самый пример 2, который демонстрировал недостатки покрытия ветвей и покрытия решений/условий, покажет и недостаток комбинаторного покрытия условий.

Критерии белого ящика требуют знания текста программы. Текста программы, анализирующей треугольники, у нас нет. Тем не менее предлагается еще раз вернуться к сформированному ранее набору тестов. Не возникнет ли желания его модифицировать?

Критерий покрытия решений/условий не гарантирует проверки такой ситуации.

Задание 2.

1. Спроектировать тесты по принципу «белого ящика» Выбрать несколько алгоритмов для тестирования и обозначить буквами или цифрами ветви этих алгоритмов.
2. Выписать пути алгоритма, которые должны быть проверены тестами для выбранного метода тестирования.
3. Записать тесты, которые позволят пройти по путям алгоритма.
4. Протестировать разработанную вами программу. Результаты оформить в виде таблиц (см. табл.1 - 4).
5. Проверить все виды тестов и сделать выводы об их эффективности.

Вопросы для закрепления теоретического материала к практическому занятию:

1. Охарактеризуйте этап реализации и тестирования программного продукта.
2. Какие существуют виды тестирования?
3. Назовите критерии выбора тестов.
4. Перечислите свойства тестов.
5. Приведите критерии надежности программ.
6. В чем заключается оценка надежности программ?

Порядок выполнения отчета по практической работе

Отчет по практической работе должен состоять из:

1. Постановки задачи.
2. Блок-схемы программ.
3. Тестов.
4. Таблиц тестирования программы.
5. Выводов по результатам тестирования (не забывайте, что целью тестирования является обнаружение ошибок в программе).

Практическая работа № 7

Тестирование программного продукта методом «черного ящика»

Цель работы состоит в приобретении навыков методов тестирования логики программы, формализованного описания результатов тестирования и применении стандартов по составлению схем программ

Краткие теоретические и учебно-методические материалы по теме практической работы

Тестирование методом «черного ящика»

Тестирование программного обеспечения включает в себя целый комплекс действий, аналогичных последовательности процессов разработки программного обеспечения. В него входят:

- постановка задачи для теста;
- проектирование теста;
- написание тестов;
- тестирование тестов;
- выполнение тестов;
- изучение результатов тестирования.

Один из подходов проектирования тестов состоит в том, что тесты проектируются на основе внешних спецификаций программ и модулей либо спецификаций сопряжения модуля с другими модулями, программа при этом рассматривается как «черный ящик». Смысл теста заключается в том, чтобы проверить, соответствует ли программа внешним спецификациям. При этом содержание модуля не имеет значения. Такой подход получил название - **стратегия «черного ящика».**

Реализация тестирования методом **«черного ящика»** сводится к проверке всех возможных комбинаций входных данных. Невозможно протестировать программу, подавая на вход бесконечное множество значений, поэтому ограничиваются определенным набором данных. При этом исходят из максимальной отдачи теста по сравнению с затратами на его создание. Она измеряется вероятностью того, что тест выявит ошибки, если они имеются и программе. Затраты измеряются временем и стоимостью подготовки, выполнения и проверки результатов теста.

Существуют следующие критерии черного ящика:

- тестирование функций;
- тестирование классов входных данных;
- тестирование классов выходных данных;
- тестирование области допустимых значений (тестирование границ класса);
- тестирование длины набора данных;
- тестирование упорядоченности набора данных.

Критерий **тестирования функций** актуален для многофункциональных программ. Он требует подобрать такой набор тестов, чтобы был выполнен хотя бы один тест для каждой из функций, реализуемых программой.

Критерий **тестирования классов входных данных** требует классифицировать входные данные, разделить их на классы таким образом, чтобы все данные из одного класса были равнозначны с точки зрения проверки правильности программы. Считается, что если программа работает правильно на одном наборе входных данных из этого класса, то она будет правильно работать на любом другом наборе данных из этого же класса. Критерий требует выполнения хотя бы одного теста для каждого класса входных данных.

Критерий **тестирования классов выходных данных** выглядит аналогично предыдущему критерию, только проверяются не входные данные, а выходные.

Часто эти три критерия хорошо согласуются друг с другом. При применении одного из них

остальные будут удовлетворены автоматически. Если программа реализует несколько функций, то вполне естественно, что каждой из этих функций будет соответствовать свой класс входных и свой класс выходных данных. Часто существует соответствие между классами входных и выходных данных.

Рассмотрим программу для учета кадров предприятия.

Она будет иметь следующие функции:

- принять на работу,
- уволить с работы,
- перевести с одной должности на другую,
- выдать кадровую сводку.
- Классы входных данных:
- приказ о приеме,
- приказ об увольнении,
- приказ о переводе,
- заявка на кадровую сводку.

Классы выходных данных:

- запись о приеме,
- запись об увольнении,
- запись о переводе,
- кадровая сводка.

Этот пример хорошо демонстрирует соответствие между функциями, классами входных и выходных данных.

Тестирование области допустимых значений (тестирование границ класса). Если область допустимых значений переменной представляет собой простое перечисление (например, ноты, цвет, пол, диагноз и т. п.), надо проверить, что программа правильно понимает все эти значения и не принимает вместо них никаких иных значений. Например, как программа отреагирует на попытку ввести несуществующую ноту или пол.

Если класс допустимых значений представляет собой числовой диапазон, то понадобится более серьезная проверка. В этом случае выделяются:

- нормальные условия (в середине класса);
- граничные (экстремальные) условия;
- исключительные условия (выход за границу класса).

Тестирование длины набора данных можно считать частным случаем тестирования области допустимых значений. В данном случае речь пойдет о допустимом количестве элементов в наборе. Если программа последовательно обрабатывает элементы некоторого набора данных, имеет смысл проверить следующие ситуации:

- пустой набор (не содержит ни одного элемента);
- единичный набор (состоит из одного-единственного элемента);
- слишком короткий набор (если предусмотрена минимально допустимая длина)
- набор минимально возможной длины (если такая предусмотрена);
- нормальный набор (состоит из нескольких элементов);
- набор из нескольких частей (если такое возможно.
- набор максимально возможной длины (если такая предусмотрена);
- слишком длинный набор (с длиной больше максимально допустимой).

Тестирование упорядоченности входных данных важно для задач сортировки и поиска экстремумов. В этом случае имеет смысл проверить следующие ситуации (классы входных данных):

- данные не упорядочены;
- данные упорядочены в прямом порядке;

- данные упорядочены в обратном порядке;
- в наборе имеются повторяющиеся значения;
- экстремальное значение находится в середине набора;
- экстремальное значение находится в начале набора;
- экстремальное значение находится в конце набора;
- в наборе несколько совпадающих экстремальных значений.

Порядок работы над программой

1. Тестирование начинают с критериев черного ящика. Соответствующие тесты составляются *до* написания текста программы. Тесты заносятся в таблицу тестов, в которой кроме входных данных и ожидаемых результатов предусмотрена графа для реальных результатов и отметки о совпадении их с ожидаемыми.
2. Пишется текст программы.
3. Проверяется текст программы, исходя из списка ошибкоопасных конструкций и ситуаций.
4. Составляется таблица МГТ (минимально грубое тестирование).
5. Прогоняются тесты, составленные исходя из критериев черного ящика. В процессе прогона строятся трассировочные таблицы.
6. При необходимости корректируется программа. Проверяются места корректировок на ошибкоопасность.
7. После корректировки программы проводится повторное тестирование: повторяются заново все ранее прогнанные тесты.
8. Заполняются последние графы таблицы тестов и таблицу МГТ.
9. Если таблица МГТ еще неполна, добавляются тесты для покрытия незаполненных строк таблицы МГТ. Заносятся в таблицу тестов.
10. Тесты прогоняются через программу. Строятся трассировки.
11. При необходимости корректируется программа. Проверяются места корректировок на ошибкоопасность.
12. Заполняются последние графы таблицы тестов и таблица МГТ.
13. После корректировки программы проводится повторное тестирование: повторяются заново все ранее прогнанные тесты.

Задания для практического занятия:

1. Выполнить упражнения
2. Описать стратегию тестирования методом «Черного ящика»
3. Описать порядок работы над программой
4. Выписать порядок тестирования классов входных и выходных данных, областей допустимых значений, длины последовательности, упорядоченности набора данных, которые должны быть проверены тестами для выбранного метода тестирования.
5. Протестировать программу. Результаты оформить в виде таблиц (см. методическое пособие).
6. Проверить все виды тестов и сделать выводы об их эффективности.
7. Оформить отчет. Сдать и защитить работу.

Упражнения

Пусть требуется написать программу, которая считает среднее арифметическое последовательности целых чисел, заканчивающейся нулем.

Пусть программа выглядит следующим образом:

```
Program p;
var a, b, c: integer:
begin
   repeat
      read(a);
      b:= b + 1;
      c:= c + a
   until a=0;
   writeln(c/b)
end.
```

Перед началом тестирования надо сформулировать цели тестирования, выбрать критерии полноты. Зафиксируем в качестве цели: провести тестирование, полное с точки зрения критериев черного ящика и критерия МГТ (минимально грубого тестирования).

Разработка тестов должна предшествовать написанию программы и первоначально должна быть направлена на уяснение поставленной задачи. Начинают с критериев черного ящика.

Первый критерий - тестирование функций - в данном случае неинтересен, поскольку программа однофункциональная. Эту единственную функцию и будем постоянно тестировать.

Второе - тестирование классов входных и выходных данных, областей допустимых значений, длины последовательности, упорядоченности набора данных. Выделение классов входных и выходных данных в этой задаче не очевидно. Областей допустимых значений - тоже. Упорядоченность здесь не важна.

В этой задаче важна проверка длины последовательности. Поэтому проверяем в такой последовательности:

1. Пустой набор (не содержит ни одного элемента).

Пустая последовательность - это последовательность, состоящая из одного нуля. Нуль элементом последовательности не является. Количество элементов в такой последовательности считается равным нулю.

Входные данные - 0.

Выходные данные. Что должна выдавать программа в ответ на пустую последовательность? В случае пустой последовательности результатом должна быть фраза «Последовательность пуста».

Вот теперь мы можем сформулировать первый тест.

Вход: 0.

Ожидаемый выход: Последовательность пуста.

2. Единичный набор (состоит из одного-единственного элемента).

Вход: 4, 0.

Ожидаемый выход: 4.

3. Слишком короткий набор.

Для нас такого понятия нет.

4. Набор минимально возможной длины.

Для нас такого понятия нет.

5. Нормальный набор (состоит из нескольких элементов).

Для определенности возьмем набор из 3 элементов.

Вход: 1, 3, 4, 0.

Среднее арифметическое выдается как вещественное число с той точностью, которую обеспечит ваш компьютер. Для определенности, пусть будет 6 знаков после запятой. Итак:

Вход: 1, 3, 4, 0.

Ожидаемый выход: 2.666667.

6. Набор из нескольких частей (если такое возможно).
В данном случае это невозможно.

7. Набор максимально возможной длины (если такая предусмотрена).
Не предусмотрена.

8. Слишком длинный набор (с длиной больше максимально допустимой).
В данном случае это невозможно.

Итоги:

1.Построено 3 теста.

2.Уточнены несколько важных деталей (что такое пустая последовательность, как на нее реагировать, какого типа должен быть результат, с какой точностью он должен выдаваться). Тесты покрывают классы выходных данных («Последовательность пуста» и число, являющееся средним арифметическим) и классы входных (3 вида последовательностей, различающихся длиной). Запишем 3 наших теста в таблицу следующего формата:

Тест	**Вход**	**Ожидаемый результат**	**Результат «сухой прокрутки»**	**+/-**	**Результат выполнения на компьютере**	**+/-**
T1	0	Последовательность пуста				
T2	4, 0	4				
T3	1,3,4,0	2,666667				

3.Заполним только первые строки и в каждой из них - первые графы. Строки, возможно, придется добавить, исходя из критериев белого ящика и анализа ошибкоопасных мест. Две средние графы заполним по результатам безмашинного тестирования, две последние - по результатам выполнения на компьютере. В графах «+/-» будем отмечать совпадение или несовпадение реального результата с ожидаемым.

4.Переходим к составлению текста программы. Среднее арифметическое - это частное от деления суммы элементов на их количество. Значит надо ввести элементы, просуммировать их, посчитать их количество, после чего напечатать частное. Например, так:

Программа на псевдокоде	Примечание

begin
выдать начальное приветствие;
ввести 1-й элемент (tek);
while последовательность не кончена do begin
увеличить сумму (sum);
увеличить количество (kol)
end;
вывести среднее арифметическое (sum/kol)
end.
	var tek: integer;
	var sum: integer;
	var kol: integer;

Упомянутые в проекте фрагменты раскроем следующим образом:

```
Ввести 1-й элемент
  writeln('Введите, пожалуйста, 1-й элемент');
  read(tek);

Последовательность не кончена
  tek<>0

Увеличить сумму
  sum:= sum + tek;

Увеличить количество
  kol:= kol + 1;
```

Первый вариант программы будет выглядеть так:

```
program SrednArifm;
var tek, kol, sum: integer;
begin
   writeln('Я считаю среднее арифметическое' +
      'последовательности чисел, кончающейся нулем');


  writeln('Введите, пожалуйста, 1-й элемент');
  read(tek);
  while tek<>0 {последовательность не кончена} do begin
    sum:= sum + tek;  {увеличить сумму элементов}
    kol:= kol + 1;    {увеличить кол-во элементов}
  end;
  if kol=0 then writeln('Последовательность пуста')
       else writeln('Среднее арифметическое=',sum/kol)
end.
```

В данном примере показана функциональность программы, учтены моменты для удобства пользователя: минимальный интерфейс.

Для упрощения чтения текста программы могут использоваться осмысленные имена переменных, абзацные отступы (запись лесенкой), комментарии.

5.Строим таблицу минимально грубого тестирования. В этой программе всего две развилки: цикл с предусловием и ветвление. В обоих случаях используются простые условия. Значит, МГТ-таблица будет содержать всего 5 строк:

			T1	T2	T3
Y1	while tek <>0	=0			
		=1			
		>1			
Y2	if col=0	+			

		-			

Имеются 3 теста построенные, исходя из критериев черного ящика.

Проверим, как они будут работать в данной программе и как отразятся в таблице МГТ (минимально грубое тестирование). Для этого вручную выполним первый тест и заполним трассировочную таблицу. Вход первого теста состоит из единственного числа - нуль. Оно и будет введено как значение переменной tek. Занесем его в трассировочную таблицу:

tek	kol	sum
0		

Тело цикла while не выполняется ни разу. Сразу переходим на условный оператор. Равна ли переменная kol нулю? Моделью оперативной памяти в данном случае является трассировочная таблица. Переменную kol необходимо инициализировать. kol — это количество элементов последовательности. Значит, изначально, пока никакой последовательности еще нет, kol должна быть равна нулю. Вставим соответствующий оператор перед вводом первого элемента последовательности. Тело программы приобретет вид:

```
begin
  writeln('Я считаю …');
  kol:= 0;
  writeln('Введите, пожалуйста, 1-й элемент');
  read(tek);
  while tek<>0 do begin
     sum:= sum + tek;
     kol:= kol + 1;
  end;
  if kol=0 then writeln('Последовательность пуста')
        else writeln('Среднее арифметическое = ', sum/kol)
end.
```

Повторим первый тест. Теперь его удастся выполнить до конца. Трассировочная таблица будет иметь вид:

tek	kol	sum
	0	
0		

В таблице тестов отметим результат выполнения первого теста:

Тест	Вход	Ожидаемый результат	Результат «сухой прокрутки»	+/-	Результат выполнения на компью-тере	+/-
T1	0	Последовательность пуста		+		
T2	4, 0	4				
T3	1,3,4,0	2,666667				

В таблице МГТ отметим проверенные строки:

			T1	T2	T3
Y1	while tek <>0	=0	+		
		=1			
		>1			
Y2	if col=0	+	+		
		-			

Перейдем ко второму тесту. Выполняем программу и заполняем трассировочную таблицу.

tek	kol	sum
	0	
4		

После входа в цикл обнаруживаем, что переменная sum не имеет начального значения. Ее, так же как и переменную kol, необходимо инициализировать нулем перед входом в цикл. Получаем следующий текст тела программы:

```
begin
   writeln('Я считаю …');
   kol:= 0;
   sum:= 0;
   writeln('Введите, пожалуйста, 1-й элемент');
   read(tek);
   while tek<>0 do begin
      sum:= sum + tek;
      kol:= kol + 1;
   end;
   if kol=0 then writeln('Последовательность пуста')
          else writeln('Среднее арифметическое=',sum/kol)
end.
```

Результаты выполнения второго теста.

tek	kol	sum
	0	
		0
4		
		4
	1	
		8
	2	
		12
	3	

Воспользуемся перечнем ошибкоопасных ситуаций. Возникли проблемы с циклом: произошло зацикливание. Чтобы его избежать, в теле цикла, как минимум, должна меняться хотя бы одна переменная, входящая в условие цикла. Такая переменная всего одна, и она в теле цикла не меняется. Необходимо ввести остальные элементы последовательности. Тело программы приобретет вид:

```
begin
  writeln('Я считаю …');
  kol:= 0;
  sum:= 0;
  writeln('Введите, пожалуйста, 1-й элемент');
  read(tek);
  while tek<>0 do begin
    sum:= sum + tek;
    kol:= kol + 1;
    writeln('Введите, пожалуйста, элемент № ',kol);
    read(tek);
  end;
  if kol=0 then writeln('Последовательность пуста')
    else writeln('Среднее арифметическое = ',sum/kol)
end.
```

Выполняем еще раз второй тест и доходим до конца.

tek	kol	sum
	0	
		0
4		
		4
	1	
0		

По ходу выполнения выясняется, что допущена еще одна небольшая ошибка в интерфейсе. Программа второй раз попросила ввести элемент № 1. Приглашение в теле цикла должно выглядеть так:

```
writeln('Введите, пожалуйста, элемент № ',kol+1);
```

Внесем в текст соответствующие изменения, на результатах выполнения это не скажется.

Сделаем отметку в таблице тестов.

Тест	Вход	Ожидаемый результат	Результат «сухой прокрутки»	+/−	Результат выполнения на компьютере	+/−
Т1	0	Последователь-ность пуста	Последователь-ность пуста	+		
Т2	4, 0	4	4	+		
Т3	1, 3, 4, 0	2.666667				

Заполним соответствующий столбец в таблице МГТ

			Т1	Т2	Т3
У1	while tek<>0	=0	+		
		=1		+	
		>1			
У2	if kol=0	+	+		
		−		+	

Так как в программу были внесены исправления, необходимо повторить все ранее прогнанные тесты.

При его повторе результаты получим прежние, хотя трассировочная таблица несколько изменится:

tek	kol	sum
	0	
		0
0		

Выполним тест 3. Строим трассировку.

tek	kol	sum
	0	
		0
1		
		1
	1	
3		
		4
	2	
4		
		8
	3	
0		

Заполняем таблицы тестов и МГТ.

Тест	Вход	Ожидаемый результат	Результат «сухой прокрутки»	+/−	Результат выполнения на компьютере	+/−
T1	0	Последователь-ность пуста	Последователь-ность пуста	+		
T2	4, 0	4	4	+		
T3	1, 3, 4, 0	2.666667	2.666667	+		

			T1	T2	T3
У1	while tek<>0	=0	+		
		=1		+	
		>1			+
У2	if kol=0	+	+		
		-		+	+

Третий тест оказался неудачен - никакой ошибки не выявил. Но все строки таблицы МГТ уже заполнены. Значит, был построен и выполнен прогон набора тестов, полный и с точки зрения критериев черного ящика, и с точки зрения минимально грубого тестирования.

Ручное тестирование выполнено.

Теперь можно выходить на компьютер и прогнать еще раз те же тесты и заполнить крайние правые графы в таблице тестов:

Тест	Вход	Ожидаемый результат	Результат «сухой прокрутки»	+/-	Результат выполнения на компьютере	+/-
T1	0	Последовательность пуста	Последовательность пуста	+	Последовательность пуста	+
T2	4, 0	4	4	+	4	+
T3	1,3,4,0	2,666667	2,666667	+	2,666667	+

В итоге сделаем 3 замечания:

1. Тестов, построенных исходя из критериев черного ящика, оказалось достаточно, чтобы удовлетворить критерию МГТ.
2. При этом тесты эти (построенные в тот момент, когда текста программы еще не существовало) оказались настолько хороши, что первые 2 из них обнаружили в нашей программе 3 ошибки. И только последний третий тест оказался неудачен, ошибок не выявил.
3. Легко заметить, что все допущенные нами ошибки упоминались в перечне ошибкоопасных ситуаций. Значит, если бы вместо того, чтобы сразу же начинать тестовые прогоны, сначала проанализировали программу с точки зрения этого перечня, ошибки удалось бы обнаружить гораздо быстрей и с меньшими усилиями.

Вопросы для закрепления теоретического материала к практическому занятию:

1. Какие виды ошибок существуют?
2. Что такое тест? Какими свойствами должен обладать тест?
3. Дайте краткую характеристику методики тестирования «черным ящиком».
4. Перечислите свойства тестов.

5. Перечислите последовательность работы с программой.

Порядок выполнения отчета по практической работе

Отчет по практической работе должен состоять из:

1. Постановки задачи.
2. Фрагментов программ.
3. Тестов.
4. Таблиц тестирования программы.
5. Выводов по результатам тестирования (не забывайте, что целью тестирования является обнаружение ошибок в программе).

Практическая работа № 8

Автоматизированное тестирование

Цель работы состоит в приобретении навыков методики автоматизированного тестирования логики программы, формализованного описания результатов тестирования и применению стандартов по составлению схем программ

Краткие теоретические и учебно-методические материалы по теме практической работы

Автоматизированное тестирование программного обеспечения (**Software Automation Testing**) - это процесс верификации программного обеспечения, при котором основные функции и шаги теста, такие как запуск, инициализация, выполнение, анализ и выдача результата, выполняются автоматически при помощи инструментов для автоматизированного тестирования.

Специалист по автоматизированному тестированию программного обеспечения (**Software Automation Tester**) - это технический специалист (тестировщик или разработчик программного обеспечения), обеспечивающий создание, отладку и поддержку работоспособного состояния тест скриптов, тестовых наборов и инструментов для автоматизированного тестирования.

Инструмент для автоматизированного тестирования (**Automation Test Tool**) - это программное обеспечение, по средствам которого специалист по автоматизированному тестированию осуществляет создание, отладку, выполнение и анализ результатов прогона тест скриптов.

Тест Скрипт (**Test Script**) - это набор инструкций, для автоматической проверки определенной части программного обеспечения.

Тестовый набор (**Test Suite**) - это комбинация тест скриптов, для проверки определенной части программного обеспечения, объединенной общей функциональностью или целями, преследуемыми запуском данного набора.

Тесты для запуска (**Test Run**) - это комбинация тест скриптов или тестовых наборов для последующего совместного запуска (последовательного или параллельного, в зависимости от преследуемых целей и возможностей инструмента для автоматизированного тестирования).

Для оценки качества ПО всегда применяется целый комплекс мер, среди которых тестирование ПО на предмет обнаружения ошибок - один из важнейших этапов.

1. Для его проведения необходимы объект тестирования - в данном случае ПО - и эталон, с которым этот объект сравнивается. Тестирование ПО проводится на соответствие заранее определенным требованиям (по функциональности, производительности, безопасности и пр.). Поскольку объект тестирования сложный, необходим системный подход к тестированию, его организации и проведению.

Требования к ПО подразделяются на функциональные (какие функции и с каким качеством

должно реализовывать ПО) и нефункциональные (ограничения на время решения задачи, скорость доступа к данным, требования к занимаемым ресурсам и т. п.). У заказчика и разработчика должна быть возможность сравнить текущее функционирование системы с ее эталонным (ожидаемым) поведением. При этом рекомендуется использовать итеративный подход, так как раннее тестирование критичных областей значительно снижает риск неудачи и стоимость исправлений для всего проекта разработки ПО.

2. Эффективнее поэтапно осуществлять контроль над ходом отработки ПО. Рекомендуется использовать шаблоны документов, в том числе плана тестирования, разработанный в соответствии с требованиями международного стандарта IEEE 829-1983 Standard for Software Test Documentation.

Обсуждение технического задания, технического проекта, архитектуры системы с заказчиком - это тоже часть процесса обнаружения ошибок и уточнения эталона.

Принято разделять тестирование по уровням задач и объектов на разных стадиях и этапах разработки ПО (см. таблицу):

- тестирование частей ПО (модулей, компонентов) с целью проверки правильности реализации алгоритмов -- выполняется разработчиками;
- функциональное тестирование подсистем и ПО в целом с целью проверки степени выполнения функциональных требований к ПО - рекомендуется проводить отдельной группой тестировщиков, не подчиненной руководителю разработки;
- нагрузочное тестирование (в том числе стрессовое) для выявления характеристик функционирования ПО при изменении нагрузки (интенсивности обращений к нему, наполнения базы данных и т. п.) - для выполнения этой работы требуются высококвалифицированные тестировщики и дорогостоящие средства автоматизации экспериментов.

Этапы тестирования

Вид тестирования	**Стадия, этап**	**Объект**	**Критерий**
Структурное, надежности	Разработка	Компоненты	Покрытие ветвлений, функции
Сборочное	Разработка	Подсистемы	Функциональность, степень проверки компонентов
Функциональное	Разработка	Система в целом	Соответствие функциональным требованиям ТЗ
Регрессионное	Разработка, сопровождение	Система в целом	Проверка качества внесения изменений
Нагрузочное	Разработка, сопровождение	Система в целом	Оценка статистических характеристик системы, соответствие ТЗ, ТТХ, подбор конфигурации оборудования
Стрессовое	Разработка, сопровождение	Система в целом	Корректность работы системы при предельных нагрузках

3. Для того чтобы увеличить объем проверок и повысить качество тестирования, обеспечить возможность повторного использования тестов при внесении изменений в ПО применяют средства автоматизации тестирования. К их числу относятся средства автоматизации функционального и нагрузочного тестирования клиент-серверных и Web-приложений: Rational TestStudio, Mercury LoadRunner, Segue SilkPerformer, а также менее популярные продукты фирм Compuware, CA, Empirix, RadView Software и др.

Тестированием надо заниматься не только постоянно, но и систематично. Если не забывать, что это процесс обнаружения ошибок, то стоит потребовать от разработчика, чтобы он периодически силами специально созданных групп проводил так называемые "review", или "структурные просмотры" проектных материалов и аудит исходных кодов программ. Заказчик вправе договориться с разработчиком о предъявлении подобных материалов или о не очень глубоком контроле хода такого процесса. Он заинтересован в том, чтобы в организации разработчика были поставлены процессы. В этом случае заказчик может быть уверен, что качество разрабатываемого ПО контролируется и обеспечивается в ходе разработки. Именно на это направлены известная модель технологической зрелости CMM (Capability Maturity Model,

www.sei.cmu.edu/cmmi/orgdocs/conops.html) и стандарт ISO 15504.

Желательно, чтобы на этапах сборки, комплексной отладки и опытной эксплуатации разработчик фиксировал интенсивность обнаружения ошибок, тогда по характеру изменения этой интенсивности можно будет судить об изменении качества ПО и, например, о целесообразности его передачи в опытную или постоянную эксплуатацию. Наконец, необходимо проведение комплекса испытаний ПО на соответствие требованиям ТЗ или других нормативных документов, на возможность эффективно работать с ПО на основе использования программной документации, приемосдаточных и других видов испытаний, обеспечивающих заказчику уверенность в работоспособности созданного для него ПО.

4.Между испытаниями, когда ПО еще только создается очень важно, чтобы деятельность по тестированию велась планомерно. Это значит, что на каждом этапе работ должны быть выбраны критерий качества и критерий завершения тестирования. Первый нужен для того, чтобы тестировщик или группа тестировщиков понимали, что и на соответствие чему они проверяют. То есть, каковы объект и эталон и какие свойства объекта проверяются. Второй критерий помогает принять решение в случае, когда исчерпываются ресурсы, отведенные на тестирование, он отвечает на вопрос, при каких условиях тестирование может быть завершено.

Для проверки сложного объекта можно и нужно применять разные стратегии, позволяющие добиваться максимального результата при существующих ограничениях на ресурсы тестирования.

План тестирования определяется международным стандартом IEEE 829-1983. В нем должны быть предусмотрены как минимум три раздела содержащие, следующие описания:

- что будет тестироваться (тестовые требования, тестовые варианты);
- какими методами и насколько подробно будет тестироваться система;
- план-график работ и требуемые ресурсы (персонал, техника).

Дополнительно описываются критерии удачного/неудачного завершения тестов, критерии окончания тестирования, риски, непредвиденные ситуации, приводятся ссылки на соответствующие разделы в основных документах проекта - план управления требованиями, план конфигурационного управления и т. п.

5.Как подготовиться к тестированию, что именно нужно для его проведения?

Необходимы проверяемые требования, затем каждое из них связывается с одним или несколькими тестовыми требованиями. Далее логический набор тестовых требований группируется в тестовые сценарии, проверка которых позволит дать односложный ответ на вопрос, корректно ли осуществляется ввод, правильно ли работает та или иная бизнес-функция в системе. Этот результат понятен не только специалистам, но и конечному пользователю. Основные объекты автоматизации тестирования - системы, реализующие работу клиентской части. Ключевой особенностью тестирования клиент-серверных систем является возможность проверки корректности функционирования и удовлетворительного быстродействия системы через работу клиентской части. Таким образом, тщательно и всесторонне проверив эти возможности, мы получаем гарантию работоспособности системы для конечного пользователя.

Еще один важный аспект организации работ - хранение созданных тестов. Рекомендуется относиться к тестам так же, как и к исходному коду, т. е. нужно использовать версионные хранилища для возможности восстановления тестов предыдущих версий системы (MS SourceSafe, Rational ClearCase,...). Это пригодится на этапе сопровождения ПО и даст возможность повторного использования готовых тестов на нескольких версиях системы. Аналогично нужно относиться и к тестовым данным, создавая архивы, копии и версии содержимого БД.

6.Тестирование - это всегда эксперимент. Для его проведения нужна база. Как в любом эксперименте, при тестировании нужно где-то собирать накопленную информацию, обрабатывать результаты. Есть самое крупное разделение видов тестирования: статическое и динамическое. При статическом тестировании ПО не исполняется, а происходит анализ кода, структур данных. Динамическое тестирование, напротив, требует выполнения тестируемого ПО. Для этого нужны не только средства автоматизации тестирования, но и вспомогательные средства. Известно очень

много средства построения и автоматической генерации тестов, средств мониторинга ресурсов во время выполнения тестов, средств измерения и визуализации результатов тестирования, средств статистической обработки результатов и т. п.

Нагрузочное тестирование

Нагрузочное тестирование (Load Testing) или **тестирование производительности** (Performance Testing) - это **автоматизированное тестирование**, имитирующее работу определенного количества бизнес пользователей на каком либо общем (разделяемом ими) ресурсе. Современное программное обеспечение просто обязано бесперебойно работать под колоссальными нагрузками. Любого рода проблемы, связанные с плохой производительностью, могут стать причиной отказа клиентов от использования вашего ПО. В связи с этим, проведение качественного нагрузочного тестирования должно стать обязательным, для обеспечения стабильности работы ваших приложений.

Начиная работу в области нагрузочного тестирования, следует четко понимать, что это не просто запись и прогон (Record and Playback) скриптов, а более сложный процесс:

Во-первых, нагрузочное тестирование - это серьезная исследовательская и аналитическая работа

Во-вторых - это реальное автоматизированное тестирование, требующее серьезных навыков программирования, а также знания сетевых протоколов и различных серверов приложений и баз данных

В-третьих - существуют разные **виды нагрузочного тестирования,** ставящие перед собой разные цели

В качестве примера можно привести работу сотрудников современного банка, в котором все работают с одними и теми же программными приложениями, установленными на банковских серверах. Или использование программного приложения веб магазин, в данном случае посетителями, нагружающими сервера, будут пользователи интернета.

Моделирование нагрузки происходит с помощью специальных продуктов и техник.

Терминология:

Виртуальный пользователь *(Virtual User)* - программный процесс, циклически выполняющий моделируемые операции

Итерация *(Iteration)* – это один повтор выполняемой в цикле операции

Интенсивность выполнения операции *(Operation Intensity)* - частота выполнения операции в единицу времени, в тестовом скрипте задается интервалом времени между итерациями

Нагрузка *(Loading)* - совокупное выполнение операций на общем ресурсе (тр./сек, хитов/сек)

Производительность *(Performance*) - количество выполняемых операций за период времени (N операций за M часов)

Масштабируемость приложения *(Application Scalability)* - пропорциональный рост производительности при увеличении нагрузки

Профиль нагрузки *(Performance Profile)* - это набор операций с заданными интенсивностями, полученный на основе сбора статистических данных либо определенный путем анализа требований к тестируемой системе

Нагрузочной точкой называется рассчитанное (либо заданное Заказчиком) количество виртуальных пользователей в группах, выполняющих операции с определенными интенсивностями

Теперь рассмотрим как эти сущности связаны между собой. Выразив интенсивность через интервал времени между итерациями, видим что рост интенсивности выполняемых операций это сокращение интервала времени. Рост нагрузки пропорционален росту интенсивности. Естественно также, что при увеличении интенсивности растет производительность. При этом увеличивается степень использования (загруженности) ресурсов. С какого-то момента рост производительности прекращается (а нагрузка может продолжать расти), происходит насыщение и затем деградация системы. В дополнение можно заметить что при тестировании изменение интенсивности операций

может подчиняться какому либо закону (например, Пуассона) либо быть равномерным в течении всего теста.

Этапы проведения нагрузочного тестирования

- Анализ требований и сбор информации о тестируемой системе.
- Разработка модели нагрузки
- Выбор инструмента для нагрузочного тестирования
- Создание и отладка тестовых скриптов
- Проведение тестирования
- Анализ результатов
- Подготовка, отправка и публикация отчета по проведенному нагрузочному тестированию

Разработка модели нагрузки

Определившись с видами нагрузочного тестирования, целями и терминологией, у вас появился определенный фундамент, чтобы узнать, что основная задача в нагрузочном тестировании - разработка модели нагрузки.

Для решения этой задачи необходимо определить следующее:

- список тестируемых операций
- интенсивность выполнения операций
- зависимость изменения интенсивности выполнения операций от времени

В список таких задач должны войти операции, которые критичны с точки зрения бизнеса и с технической точки зрения. Критичностью с точки зрения бизнеса (и это основной критерий) является реальное влияние ухудшения производительности таких операций на бизнес процессы. Например, увеличение длительности обслуживания клиентов в банке, невозможность выполнить необходимое количество операций в течение дня и так далее. С технической точки зрения - это операции максимально потребляющие ресурсы серверов. Как правило - это операции выполняемые большим количество бизнес пользователей одновременно или создание сложных отчетов, в которые входят так называемые «тяжелые» запросы к базе данных. Хотим еще раз подчеркнуть, что степенью критичности является влияние на бизнес и работоспособность системы, например, создание отчета полностью загружающего сервер базы данных, в ночное время, не будет носить высокий приоритет для оптимизации, а в рабочие часы это будет иметь максимальный приоритет.

Изучение Приложения (тестируемое прикладное программное обеспечение)

Чтобы выделить части приложения, а именно операции, которые будут тестироваться, необходимо провести работу связанную с изучением приложения. Очень большую пользу при этом должны оказать разработчики приложения, если речь идет о тестировании в процессе разработки, либо бизнес пользователи и системные администраторы, если приложение находится в процессе эксплуатации. В ходе этой работы разумно сделать такие шаги:

Описать компоненты приложения и составить схемы взаимодействия между ними

Выделить критические с точки зрения предполагаемого тестирования операции. В качестве таковых могут быть выбраны:

- Операции с «тяжелыми» запросами к базе данных, процессы генерации отчетов
- Операции, выполняемые большим количеством пользователей или с высокой интенсивностью
- Операции критичные с точки зрения бизнеса, и к тому же удовлетворяющие условиям двух верхних пунктов

Еще раз хочется заметить, что опрос бизнес пользователей или совместное исследование с

разработчиками и администраторами системы может значительно облегчить задачу. Если приложение находится в эксплуатации, то можно провести мониторинг загрузки компонентов аппаратных серверов (процессора, память, диски) и проанализировать системные журналы веб серверов (снять stats pack, если в качестве сервера базы данных, например, используется Oracle). Системные журналы могут показать пики высокой активности пользователей в течение дня и дать количественное оценки того сколько транзакций (хитов) выполняется в единицу времени. Согласно закону Паретто или принципу 20/80, 20% операций приложения генерируют 80% нагрузки в системе, поэтому нужно стараться выбрать для моделирования именно эти 20% операций.

Определение профиля нагрузки

Ключевым моментом в модели нагрузки являются выбранные для тестирования операции или профиль нагрузки. Естественно выполняться эти операции в тесте должны одновременно. Профилей нагрузки для приложения может быть несколько и это оправдано. Ведь бизнес пользователи могут выполнять разные наборы операций в разное время. Например, начало операционного дня и конец дня, начало месяца (квартала) и соответственно завершение могут отличаться. Таким образом получаем различные наборы операций приложения, выполняющиеся одновременно и соответственно создающие различную нагрузку. Меняться могут не только сами операции но и их интенсивности. В первом приближении моделью нагрузки является набор профилей нагрузки, где каждый профиль отличается от другого или набором операций или интенсивностями выполнения этих операций.

Пример профиля нагрузки, в который входит 5 операций, значение n может быть различным для каждой операции:

<Профиль нагрузки>
Операция_1 - интенсивность выполнения n раз / ед. времени
Операция_2 - интенсивность выполнения n раз / ед. времени
Операция_3 - интенсивность выполнения n раз / ед. времени
Операция_4 - интенсивность выполнения n раз / ед. времени
Операция_5 - интенсивность выполнения n раз / ед. времени

Расчет нагрузочных точек

Поскольку в профиле нагрузки как правило присутствует несколько операций - это означает, что у нас будет несколько групп пользователей. Желательно моделировать каждую операцию отдельной группой виртуальных пользователей (хотя в жизни часто бывает наоборот, один бизнес пользователь может отвечать за выполнение нескольких операций). Тем не менее, если назначить одному виртуальному пользователю выполнение одной операции, то так легче выдержать определенную интенсивность (и соответственно производительность) для этой операции в тесте, чем в случае, когда виртуальному пользователю назначается последовательная цепочка операций. Зная интенсивность выполнения операции нужно определить количество виртуальных пользователей в группе, выполняющих эту операцию. Идеальный случай, когда работа с приложением аналогична работе заводского конвейера и есть точные оценки сколько операций в день делает один пользователь. Чаще всего бывает не так и известно только общее количество операций выполняемое в течение дня. Так же может оказаться, что интенсивность выполнения операции каждым пользователем очень низкая, например, один пользователь выполняет операцию раз в день или раз в два дня.

Автоматизированное тестирование использует программные средства для выполнения тестов и проверки результатов выполнения, что помогает сократить время тестирования и упростить его процесс.

Существует два основных подхода к автоматизации тестирования: тестирование на уровне кода и GUI-тестирование. К первому типу относится, в частности, модульное тестирование. Ко второму - имитация действий пользователя с помощью специальных тестовых «фреймворков».

Наиболее распространенной формой автоматизации является тестирование приложений через графический пользовательский интерфейс. Популярность такого вида тестирования объясняется двумя факторами: во-первых, приложение тестируется тем же способом, которым его будет использовать человек, во-вторых, можно тестировать приложение, не имея при этом доступа к исходному коду.

Одной из главных проблем автоматизированного тестирования является его трудоемкость: несмотря на то, что оно позволяет устранить часть рутинных операций и ускорить выполнение тестов, большие ресурсы могут тратиться на обновление самих тестов. Это относится к обоим видам автоматизации. При рефакторинге часто бывает необходимо обновить и модульные тесты, и изменение кода тестов может занять столько же времени, сколько и изменение основного кода. С другой стороны, при изменении интерфейса приложения необходимо заново переписать все тесты, которые связаны с обновленными окнами, что при большом количестве тестов может отнять значительные ресурсы.

Для автоматизации тестирования существует большое количество приложений. Наиболее популярные из них (по итогам 2007 года):

- HP LoadRunner, HP QuickTest Professional, HP Quality Center
- Segue SilkPerformer
- IBM Rational FunctionalTester, IBM Rational PerformanceTester, IBM Rational TestStudio
- AutomatedQA TestComplete

Использование этих инструментов помогает тестировщикам автоматизировать следующие задачи:

- установка продукта
- создание тестовых данных
- GUI взаимодействие
- определение проблемы

Однако автоматические тесты не могут полностью заменить ручное тестирование. Автоматизация всех испытаний — очень дорогой процесс, и потому автоматическое тестирование является лишь дополнением ручного тестирования. Наилучший вариант использования автоматических тестов - регрессионное тестирование.

Сравнение ручного и автоматизированного тестирования

Результаты сравнения приведены в таблице 1. Сравнение показывает тенденцию современного тестирования, ориентирующую на максимальную автоматизацию процесса тестирования и генерацию тестового кода, что позволяет справляться с большими объемами данных и тестов, необходимых для обеспечения качества при производстве программных продуктов.

Таблица 1. Сравнение ручного и автоматизированного подхода

	Ручное	Автоматизированное
Задание входных значений	Гибкость в задании данных. Позволяет использовать разные значения на разных циклах прогона тестов, расширяя покрытие	Входные значения строго заданы
Проверка результата	Гибкая, позволяет тестировщику оценивать нечетко сформулированные критерии	Строгая. Нечетко сформулированные критерии могут быть проверены только путем сравнения с эталоном
Повторяемость	Низкая. Человеческий фактор и нечеткое определение данных приводят к неповторяемости тестирования	Высокая
Надежность	Низкая. Длительные тестовые циклы приводят к снижению внимания тестировщика	Высокая, не зависит от длины тестового цикла
Чувствительность к незначительным изменениям в продукте	Зависит от детальности описания процедуры. Обычно тестировщик в состоянии выполнить тест, если внешний вид продукта и текст сообщений несколько изменились	Высокая. Незначительные изменения в интерфейсе часто ведут к корректировке эталонов
Скорость выполнения тестового набора	Низкая	Высокая
Возможность генерации тестов	Отсутствует. Низкая скорость выполнения обычно не позволяет исполнить сгенерированный набор тестов	Поддерживается

Тестовый отчет

Тестовый отчет обновляется после каждого цикла тестирования и должен содержать следующую информацию для каждого цикла:

1. Перечень функциональности в соответствии с пунктами требований, запланированный для тестирования на данном цикле, и реальные данные по нему.
2. Количество выполненных тестов – запланированное и реально исполненное.
3. Время, затраченное на тестирование каждой функции, и общее время тестирования.
4. Количество найденных дефектов.
5. Количество повторно открытых дефектов.
6. Отклонения от запланированной последовательности действий, если таковые имели место.
7. Выводы о необходимых корректировках в системе тестов, которые должны быть сделаны до следующего тестового цикла.

Оценка качества тестов

Тесты нуждаются в контроле качества так же, как и тестируемый продукт. Поскольку тесты для продукта являются своего рода эталоном его структурных и поведенческих характеристик, закономерен вопрос о том, насколько адекватен эталон. Для оценки качества тестов используются различные методы, наиболее популярные из которых кратко рассмотрены ниже.

Тестовые метрики

Существует устоявшийся набор тестовых метрик, который помогает определить эффективность тестирования и текущее состояние продукта. К таким метрикам относятся следующие:

1. Покрытие функциональных требований.
2. Покрытие кода продукта. Наиболее применимо для модульного уровня тестирования.
3. Покрытие множества сценариев.
4. Количество или плотность найденных дефектов. Текущее количество дефектов сравнивается со средним для данного типа продуктов с целью установить, находится ли оно в пределах допустимого статистического отклонения. При этом обнаруженные отклонения как в большую, так и в меньшую сторону приводят к анализу причин их появления и, если необходимо, к выработке корректирующих действий.
5. Соотношение количества найденных дефектов с количеством тестов на данную функцию продукта. Сильное расхождение этих двух величин говорит либо о неэффективности тестов (когда большое количество тестов находит мало дефектов) либо о плохом качестве данного участка кода (когда найдено большое количество дефектов на не очень большом количестве тестов).
6. Количество найденных дефектов, соотнесенное по времени, или скорость поиска дефектов. Если производная такой функции близка к нулю, то продукт обладает качеством, достаточным для окончания тестирования и поставки заказчику.

Обзоры тестов и стратегии

Тестовый код и стратегия тестирования, зафиксированные в виде документов, заметно улучшаются, если подвергаются коллективному обсуждению. Такие обсуждения называются обзорами (review). Существует принятая в организации процедура проведения и оценки результатов обзора. Обзоры наряду с тестированием образуют мощный набор методов борьбы с ошибками с целью повышения качества продукта. Цели обзоров тестовой стратегии и тестового кода различны.

Цели обзора тестовой стратегии:

1. Установить достаточность проверок, обеспечиваемых тестированием.
2. Проанализировать оптимальность покрытия или адекватность распределения количества планируемых тестов по функциональности продукта.
3. Проанализировать оптимальность подхода к разработке кода, генерации кода, автоматизации тестирования.

Задания для практического занятия:

- Протестировать разработанный программный продукт (по вариантам).
- Написать отчет.

Вопросы для закрепления теоретического материала к практическому занятию:

1. Что такое автоматизированное тестирование?
2. Чем отличается автоматизированное тестирование от «ручного»?
3. Тестовый отчет. Состав тестового отчета.
4. Два основных подхода к автоматизации тестирования.
5. Метрики. Виды метрик.

Порядок выполнения отчета по практической работе

1. Сохранить работу на диске.

2. Ответить на контрольные вопросы.

Практическая работа №9

Отладка программного продукта

Цель работы состоит в приобретении навыков применения правил и последовательности отладки программного продукта

Краткие теоретические и учебно-методические материалы по теме практической работы

Процесс исправления ошибок называется отладкой. Отладка программ и обработка ошибок всегда выступает как часть процесса разработки. В большинстве систем разработки имеются инструменты, с помощью которых можно решить проблемы, возникающие в процессе программирования. В VBA также есть средства, которые позволяют либо исключить ошибки при разработке, либо задать отклик на ошибки при выполнении программ.

Отладка программ и обработка ошибок - это не одно и то же, но они тесно связаны друг с другом.

Отладка программ - это проверка и внесение исправлений в программу при ее разработке. Отладка позволяет идентифицировать ошибки, допущенные при программировании. Например, синтаксические ошибки в тексте программы, именах функций и переменных или логические ошибки.

Обработка ошибок - это задание реакции на ошибки, которые возникают во время выполнения программы. Причиной ошибок могут быть как ошибки в самой программе, так и другие обстоятельства, находящиеся вне сферы влияния программиста. Например, отсутствие файлов, к которым происходит программное обращение, отказ аппаратных средств или неправильные действия пользователя.

Невозможно предотвратить возникновение всех ошибок, но следует стремиться к уменьшению их числа. В маленькой программе довольно просто выявить ошибку. Однако по мере увеличения размеров и сложности программ находить их становится все труднее. В таких случаях необходимо пользоваться средствами отладки VBA.

Среда разработки программ на VBA предоставляет пользователю современные удобные средства отладки программы: предположим, что уже написан код вашей процедуры. Следующий этап в создании любой процедуры - тестирование написанного кода.

Тестирование - это процесс выполнения процедуры и исследование всех аспектов ее работы.

Цель тестирования - проверить правильность результатов выполнения процедуры и ее реакцию на разнообразные действия пользователя.

Если во время работы процедуры получены неверные результаты вычислений, непредвиденная реакция на те или иные действия пользователя, либо вообще произошла остановка выполнения, то это говорит о том, что в тексте программы имеются ошибки.

Все возможные ошибки можно разделить на три вида:

1. *Ошибки компиляции*. Возникают, если VBA не может интерпретировать введенный текст, например, при использовании неправильного синтаксиса инструкции или задании неверного имени метода или свойства. Некоторые ошибки компиляции обнаруживаются при вводе инструкции, а другие - только перед выполнением программы. Данный тип ошибок обычно просто идентифицировать и исправить, поскольку VBA выявляет их автоматически, а сами ошибки очевидны.

2. *Ошибки выполнения*. Возникают при выполнении программы, т.е. после успешной компиляции. Причиной таких ошибок может быть отсутствие данных или неправильная информация (например, данные, введенные пользователем). Ошибки выполнения, как и ошибки компиляции, легко идентифицируются VBA. При этом выводится инструкция, при выполнении которой произошла ошибка. Ошибки данного типа тяжелее устранить: может понадобиться вывести значения переменных или свойств, а также другие данные, которые влияют на успешное выполнение программы.

3. *Логические ошибки* труднее всего заметить и устранить. Логические ошибки не приводят к прекращению компиляции или выполнения. Однако они являются причиной того, что программа не выдает желаемых результатов. Ошибки данного типа идентифицируются путем тщательной проверки с помощью средств отладки VBA.

Компиляция — это процесс преобразования программы, написанной на алгоритмическом языке, в язык машинных кодов. Если в программе есть синтаксические ошибки, то процесс компиляции прекращается, строки с ошибкой закрашиваются желтым цветом и выдается соответствующее сообщение. Для продолжения выполнения программы необходимо исправить ошибку и нажать кнопку «Continue» на стандартной панели редактора VBA. Или прервать выполнение программы, нажав на кнопку «Reset» , исправить ошибку в программе, а затем заново запустить ее. При обнаружении ошибки компилятор выдает сообщение с указанием номера ошибки. В этом случае полезно, воспользовавшись справочной системой редактора VBA, определить характер ошибки и исправить ее.

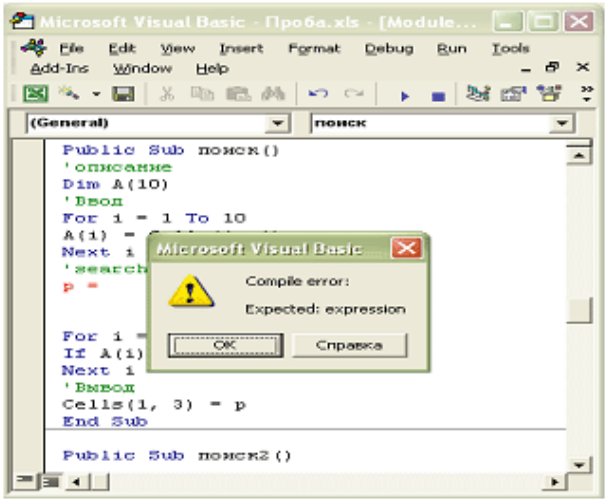


Рис. 1. Редактор Visual Basic немедленно реагирует на синтаксические ошибки

Чтобы исследовать процесс отладки на практике, нам необходима какая-нибудь программа, содержащая ошибку. В последующем примере написана такая программа «Отладка программ» рассматривается устранение ошибки при написании процедуры для объекта Image.

Задания для практического занятия:

1. Выполнить упражнения

2. Оформить работу в соответствии с ГОСТ 19.106-78. При оформлении использовать MS Office.

3. Сдать и защитить работу

Упражнения

Задание 1.

1. Откройте новую рабочую книгу.

2. Подготовьте экранную форму, представленную на рис.2. Внедрите в созданную форму с помощью панели Toolbox объект Image  . Рисунок лучше внедрить небольшой.

ВНИМАНИЕ!!! Правильно описывайте путь к графическим файлам, которые внедряются программно в форму.

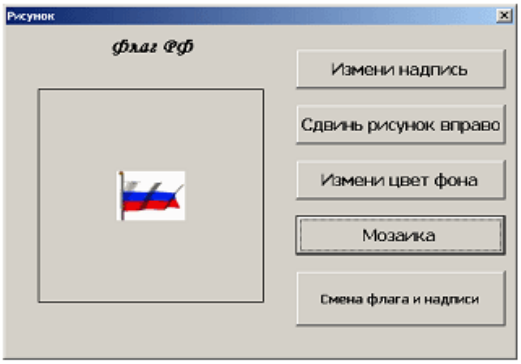


Рис. 2. Форма для выполнения задания

3. Создайте новую процедуру для кнопки «Измени надпись».

4. Введите текст процедуры. В тексте намеренно сделаем ошибку в свойстве Size (напишем Sie):

```
Private Sub CommandButton1_Click()
Label1.Caption = "Флаг России"
UserForm2.Image1.Picture = LoadPicture("C:\FlgRUS.gif")
Label1.Font.Sie = 14
End Sub
```

5. Вернемся в редакторе к созданной форме и выведем форму для работы, нажав клавишу.

6. После появления формы на экране нажмем на кнопку «Измени надпись». Так как в программе заложена ошибка, появится окно сообщения об ошибке (рис. 3), и открывается редактор VBA.

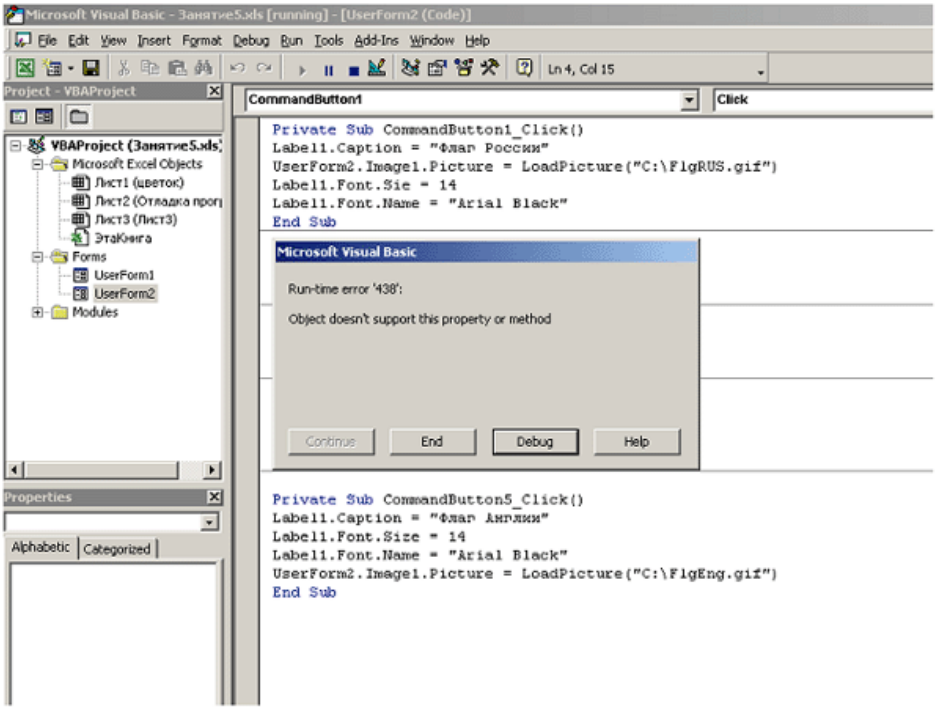


Рис. 3. Окно редактирования кода с окном сообщения об ошибке

7. Нажмите на кнопку «Debug» (отладка), и отладчик укажет, в какой строке у вас ошибка (рис. 4).

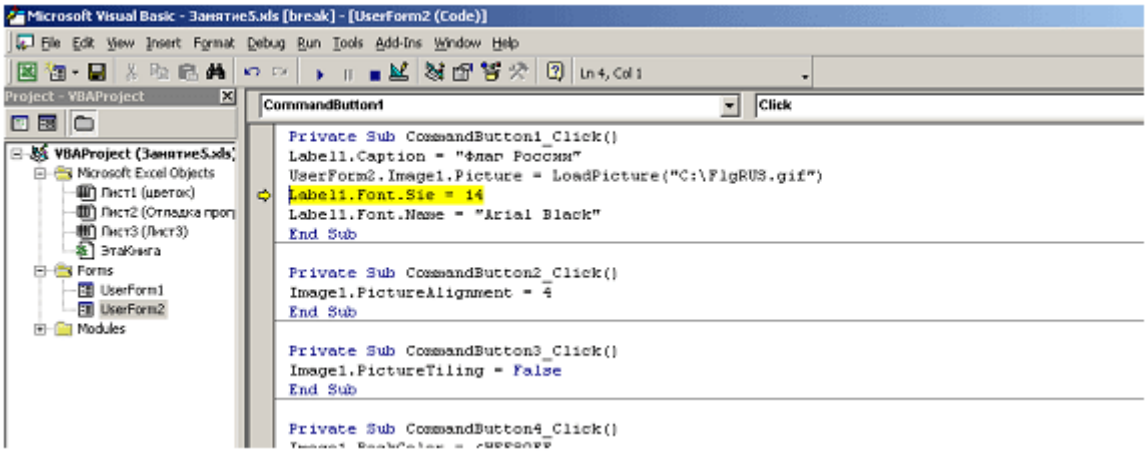


Рис. 4. Окно редактирования кода с указанной ошибкой

8. Исправьте ошибку и нажмите на стандартной панели инструментов на кнопку  («Продолжение»).

Тексты программ для кнопок CommandButton2, CommandButton3, CommandButton4, CommandButton5 представлены в таблице:

Объект	Программа
CommandButton2 (сдвинь рисунок вправо)	Private Sub CommandButton2_Click() Image1.PictureAlignment = 4 End Sub
CommandButton4 (измени цвет фона и	Private Sub CommandButton4_Click()

формы)	Image1.BackColor = &HFF80FF UserForm2.BackColor = RGB(64, 0, 0) End Sub
CommandButton3 (мозаика)	Private Sub CommandButton3_Click() Image1.PictureTiling = True End Sub
CommandButton5 (измени рисунок флага и надпись)	Private Sub CommandButton5_Click() Label1.Caption = "Флаг Англии" Label1.Font.Size = 14 Label1.Font.Name = "Arial Black" UserForm2.Image1.Picture = LoadPicture("C:\FlgEng.gif") End Sub

9. После щелчка по кнопке «Измени надпись» форма приобретет вид, представленный на рис. 5.

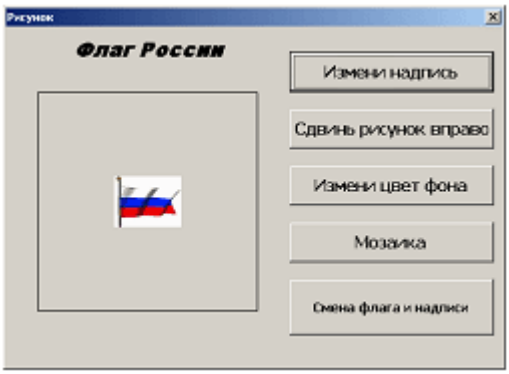


Рис. 5. Работа кнопки «Измени надпись»

10. После щелчка по кнопке «Сдвинь рисунок вправо» форма приобретет вид, представленный на рис. 6.

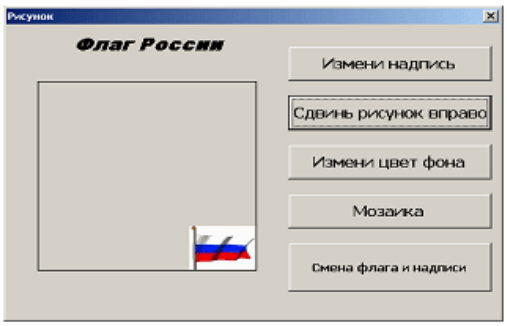


Рис. 6. Работа кнопки «Сдвинь рисунок вправо»

11. После щелчка по кнопке «Мозаика» форма приобретет вид, представленный на рис. 7.

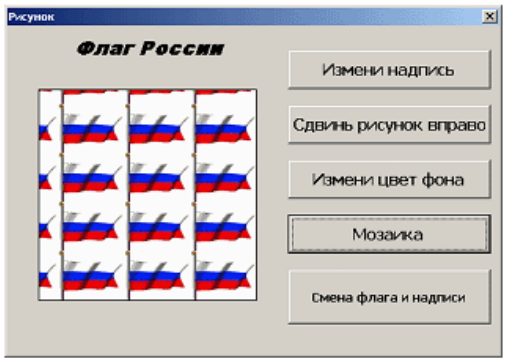


Рис. 7. Работа кнопки «Мозаика»

12. После щелчка по кнопке «Смена флага и надписи» форма приобретет вид, представленный на рис. 8.

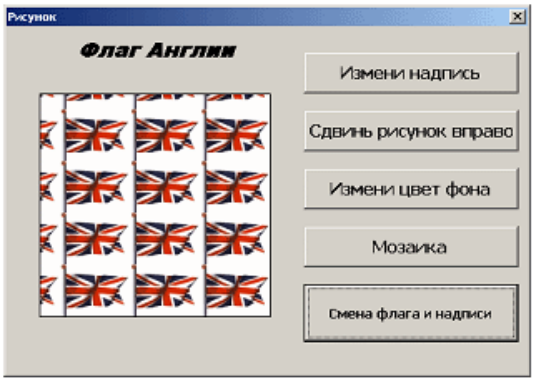


Рис. 8. Работа кнопки «Смена флага и надписи»

Можно предусмотреть разные комбинации рисунков и надписей.

13. Сохраните свою работу.

Задание 2.

1.Написать код на программный продукт с использованием редактора кода VBA, содержащий ошибку и показать преподавателю (см. пример).

2. Провести отладку программного продукта.

Вопросы для закрепления теоретического материала к практическому занятию:

1. Какие ошибки в программах существуют?
2. Что понимают под отладкой программы?
3. Чем отладка отличается от тестирования?

Порядок выполнения отчета по практической работе

Отчет по практической работе должен состоять из программного продукта, содержащего ошибку и копии программного продукта без ошибки.

Защита отчета по практической работе заключается в предъявлении преподавателю полученных результатов (на экране монитора и печатном виде), демонстрации полученных навыков и ответах на вопросы преподавателя.

Практическая работа № 10

Коллективная разработка программного продукта

Цель работы состоит в приобретении навыков работы в составе бригады при разработке программного продукта

Краткие теоретические и учебно-методические материалы по теме практической работы

В настоящее время сложность промышленных приложений и систем такова, что процесс их разработки стал практически неуправляемым. Кроме того, их развертывание на сотнях компьютеров, расположенных в разных местах, значительно раздвигает границы процесса разработки.

Один человек не способен создать приложение масштаба предприятия. Ни один разработчик просто не удержит в голове все требования к системе и варианты проекта. Поэтому сегодня разработкой промышленных систем занимаются проектные группы, и все обязанности распределяются среди членов группы.

Существует две основные модели организации коллектива при разработке ПО:

1) иерархическая модель;

2) модель группы.

Работа в коллективе отличается определенными сложностями. Иерархическая модель организации, определяет начальников и подчиненных. Однако если в современных производственных средах один менеджер проекта отвечает за все тонкости разработки и принимает все важные решения, возникает множество проблем, ведущих к провалу проекта. Опыта одного человека чаще всего недостаточно для быстрого решения задачи и для интеграции приложения в существующую инфраструктуру.

В организациях, построенных на основе иерархической модели, затруднен обмен информацией - в этой модели он, по определению, осуществляется через посредников. Вся информация иерархических групп «фильтруется» тремя или четырьмя менеджерами, что значительно повышает вероятность утери самого важного. Часто такое отсеивание идей происходит при прохождении сообщения от разработчика, непосредственно занимающегося проектом, к высшему руководству. Естественно, некоторые участники «выпадают» из процесса, что снижает эффективность их труда и повышает вероятность провала проекта.

Чтобы сгладить недостатки иерархической модели, в проектной группе предусматривается распределение обязанностей руководителя между членами коллектива. При этом за проект отвечает не один человек, а все члены группы - каждый за свой участок.

Модель группы не определяет структуру коллектива с точки зрения отдела кадров. Ведь в такую разностороннюю группу привлечены ресурсы из разных отделов организации. Задача модели проектной группы - определить цели проекта и распределить обязанности. Руководители каждого направления с помощью выделенных им ресурсов выполняют возложенную на них часть работы. Обязанности ролей определяются работой над проектом, а не деятельностью «штатной единицы». При этом руководители направлений выполняют свои обычные функции: составляют график выплаты премий, распределяют отпуска и контролируют эффективность работы сотрудников. Начальник может оценить степень участия и эффективность работы сотрудников в проектной группе, но это - прерогатива менеджера конкретного сотрудника, а не проектной группы.

Для достижения целей в модели проектной группы выполняемые задачи распределяются по шести ролям: менеджмент продукта, менеджмент программы, разработка, тестирование, обучение пользователей и логистика. Люди, выполняющие конкретную роль, должны обладать необходимой для этого квалификацией.

В этой модели нет руководителя всего проекта - есть группа людей, знающих, что нужно делать и делающих это.

Иерархическая модель.

Недостатки:

- нехватка информации;
- невозможностью учесть все особенности проекта;
- отсутствие полноценной связи между всеми участниками проекта, так как вся информация идет в одном направлении - вверх по иерархии, к главному менеджеру;
- трудность освоения новых технологий, необходимых при создании кроссплатформенных приложений;
- сложность расстановки приоритетов.

Кроме того, опыта одного человека чаще всего недостаточно для быстрого решения задачи и для интеграции приложения в существующую инфраструктуру.

В организациях, построенных на основе иерархической модели, затруднен обмен информацией - в этой модели он, по определению, осуществляется через посредников. Вся информация иерархических групп «фильтруется» тремя или четырьмя менеджерами, что значительно повышает вероятность утери самого важного. Часто такое отсеивание идей происходит при прохождении сообщения от разработчика, непосредственно занимающегося проектом, к высшему руководству. Естественно, некоторые участники «выпадают» из процесса, что снижает эффективность их труда и повышает вероятность провала проекта.

Чтобы сгладить недостатки иерархической модели, в проектной группе предусматривается распределение обязанностей руководителя между членами коллектива. При этом за проект отвечает не один человек, а все члены группы - каждый за свой участок.

Модель группы не определяет структуру коллектива с точки зрения отдела кадров. Ведь в такую разностороннюю группу привлечены ресурсы из разных отделов организации. Задача модели проектной группы - определить цели проекта и распределить обязанности. Руководители каждого направления с помощью выделенных им ресурсов выполняют возложенную на них часть работы. Обязанности ролей определяются работой над проектом, а не деятельностью «штатной единицы». При этом руководители направлений выполняют свои обычные функции: составляют график выплаты премий, распределяют отпуска и контролируют эффективность работы сотрудников. Начальник может оценить степень участия и эффективность работы сотрудников в проектной группе, но это - прерогатива менеджера конкретного сотрудника, а не проектной группы.

Коллективный подход.

Недостатки:

- разрозненная связь с внешними источниками информации;
- несогласованное представление о разных сторонах проекта;
- несогласованность личных планов членов группы;
- отсутствие опыта, снижающее эффективность коллективной работы.

Обязанности членов группы. Предлагается подход MFS.

MSF - не готовое решение, а каркас, который можно адаптировать для нужд любой организации. Один из элементов этого каркаса - *модель проектной группы.* Она описывает структуру группы и принципы, которым надо следовать для успешного выполнения проекта.

Хотя модель группы разработчиков весьма конкретна, при знакомстве с MSF ее нужно рассматривать в качестве отправной точки. Разные коллективы реализуют этот каркас по-разному, в зависимости от масштаба проекта, размеров группы и уровня подготовки ее членов.

Чтобы проект считался удачным, следует решить определенные задачи:

- **удовлетворить требования заказчика -** проект должен выполнить требования заказчиков и пользователей, иначе ни о каком успехе не может быть и речи, возможна ситуация, когда бюджет и график соблюдены, но проект провалился, так как не выполнены требования заказчика;
- **соблюсти ограничения -** разработчики проекта должны уложиться в финансовые и временные рамки;
- **выполнить спецификации, основанные на требованиях пользователей -** спецификации - это подробное описание продукта, создаваемое группой для заказчика; они представляют собой соглашение между проектной группой и клиентом и регулируют вопросы, касающиеся приложения, в основе этого требования лежит принцип «сделать все, что обещано»;
- **выпустить продукт только после выявления и устранения всех проблем -** не существует программ без дефектов, однако группа должна найти и устранить их до выпуска продукта в свет, причем устранением ошибки считается не только ее исправление, но и, например, занесение в документацию способа ее обхода; даже такой способ устранения проблем предпочтительнее, чем выпуск приложения, содержащего не выявленные ошибки, которые в любой момент могут преподнести неприятный сюрприз пользователям и разработчикам;
- **повысить эффективность труда пользователей -** новый продукт должен упрощать работу пользователей и делать ее более эффективной. Поэтому приложение, обладающее массой возможностей применять которые сложно или неудобно, считается провальным;
- **гарантировать простоту развертывания и управления -** эффективность развертывания непосредственно влияет на оценку пользователем качества продукта, например, ошибка в программе установки может создать у пользователей впечатление, что и само приложение небезгрешно, от проектной группы требуется не только подготовить продукт к развертыванию и гладко провести его, но и обеспечить пользователей поддержкой, организовав сопровождение приложения.

Для достижения этих целей в модели проектной группы выполняемые задачи распределяются по шести ролям: менеджмент продукта, менеджмент программы, разработка, тестирование, обучение пользователей и логистика. Люди, выполняющие конкретную роль, должны обладать необходимой для этого квалификацией.

Шесть ролей модели проектной группы проиллюстрированы в таблице. Все эти цели важны для успеха проекта в целом, и поэтому все роли равноправны. В этой модели нет руководителя всего проекта - есть группа людей, знающих, что нужно делать и делающих это.

Таблица - Цели и роли

Цель	Роль
Удовлетворение требований заказчика	Менеджер продукта
Соблюдение ограничений проекта	Менеджер программы
Соответствие спецификациям	Разработчик
Выпуск только после выявления и устранения проблем	Тестер
Повышение эффективности труда пользователя	Инструктор
Простота развертывания и постоянное сопровождение	Логистик

Как начать работу над проектом, не зная, сколько времени на это потребуется, сколько проект будет стоить и каких результатов ожидать? Ответить на эти вопросы поможет модель проектной группы MSF и модель процесса разработки. Обе эти модели предлагают составлять расписания и графики «снизу - вверх», в проектировании придерживаться подхода «сверху - вниз» и, кроме этого, распределять ответственность за результаты работы между членами группы. Приняв на этих стадиях правильное решение, можно избежать серьезных изменений в дальнейшем, что намного сократит время выполнения проекта и затраты на него.

Модель проектной группы

В проектную группу должны входить:

- опытные руководители;
- инициативные сотрудники, способные принимать решения и нести ответственность за свое направление работы.

Их задача:

- сконцентрироваться на выпуске продукта;
- выработать общее представление о проекте.

Для эффективной работы проектной группы требуется соблюдение следующих правил в отношениях между людьми:

- **доверие -** делает действия людей согласованными, при этом все
- следуют принципу «мы делаем то, что обещали сделать»;
- **уважение -** люди признают способности других, следуя правилу

Рисунок 1 - Роли в модели проектной группы

- «каждый из нас необходим нашей группе»;

- **согласие -** все должны знать и поддерживать цели проекта и верить в его успех - «мы завершим проект, и точка»;
- **ответственность -** люди должны ясно понимать цели проекта, свои обязанности и чего от них ожидают - «я сделаю свою работу, вы - свою, и к четвергу мы построим наш дом».

Менеджер продукта

Менеджер продукта должен вовремя реагировать на потребности заказчика. Его главная задача - сформировать общее представление о поставленной задаче и о том, как ее решать. Он должен ответить на вопрос «Зачем мы делаем все это?» и убедиться, что все члены группы знают и понимают ответ на него.

Основная цель этой роли - удовлетворение требований заказчика. Для этого менеджер продукта выступает представителем заказчика в группе разработчиков и представителем группы у заказчика. (На этом этапе важно понимать разницу между заказчиком и пользователем: заказчик платит за создание продукта, а пользователи с ним работают.) Кроме того, менеджер продукта вместе с менеджером программы должны прийти к компромиссному решению относительно функциональных возможностей продукта, сроков его разработки и финансирования проекта.

Менеджер программы

Задача менеджера программы - вести процесс разработки с учетом всех ограничений. Руководитель этого направления должен понимать разницу между понятиями «руководитель» и «начальник». Главная обязанность менеджера программы - выполнить все стадии разработки так, чтобы нужный продукт был выпущен в нужное время. Он координирует деятельность других членов группы. И хотя иногда ему придется подгонять своих сотрудников, он не должен и помышлять о диктаторском стиле управления.

Главный менеджер программы составляет график проекта на основе информации, полученной от остальных членов группы. Он координирует этот график с руководителями всех подгрупп. Буферным временем проекта также управляет менеджер программы. Если отдельные части работы выполняются раньше графика или, наоборот, задерживаются, именно менеджер программы должен выяснить, как это скажется на проекте, и изменить график.

Менеджер программы отвечает и за бюджет проекта, объединяя требования к ресурсам всех членов группы в единый план расходов. Естественно, его задача - не только разобраться в этих требованиях, но и контролировать реальные затраты, сравнивая их с запланированными. Кроме того, менеджер программы должен регулярно сообщать о состоянии работы всем основным участникам проекта.

Разработчик

Разработчики знакомят остальных членов группы с применяемыми технологиями и собственно создают продукт. В качестве консультантов они предоставляют исходные данные для проектирования, проводят оценку технологий, а также разрабатывают прототипы и тестовые системы, необходимые для проверки решений и сокращения рисков на ранних стадиях процесса разработки. Чтобы создать продукт определенного качества, разработчикам не следует замыкаться на создании кода, они должны участвовать и в решении прикладной задачи. Они творят не ради творчества, а для реализации требований заказчика. Часто, чтобы полностью разобраться в проекте, приходится создавать прототипы, а чтобы протестировать новую технологию, - испытательные системы, помогающие принять окончательное решение относительно архитектуры приложения. Этим также занимаются разработчики.

Разработчики сами оценивают сроки своей работы. Разработчики отвечают и за техническую реализацию проекта - в основном на фазах создания логической и физической модели. На этих стадиях их задача - определить методы реализации функциональных возможностей и заданной архитектуры, а также оценить сроки выполнения этой работы. Заметим, что разработчики не выбирают функции - они только решают, как

их реализовать.

Кроме того, на стадии «Планирование» разработчики решают, какое влияние окажет на проект добавление или удаление некоторых функций. Разработчики не участвуют в заключительной стадии проекта - развертывании продукта, однако они должны тесно сотрудничать с логистиками на стадии установки приложения.

Тестер

Задача тестеров - испытание продукта в реальных условиях, дабы определить, что в продукте работает и что не работает, и нарисовать таким образом точный «портрет» приложения. Естественно, для проведения тестов нужно отлично разбираться и в требованиях пользователей, и в том, как их удовлетворить.

Тестеры разрабатывают стратегию, планы, графики и сценарии тестирования, которые позволяют убедиться, что все ошибки выявлены и исправлены до выпуска приложения. Ошибкой называют любую проблему, из-за которой продукт не выполняет свои функции. Ею может оказаться и ошибка в коде, называемая «жучком», и отклонение от спецификаций, заданных менеджером программы, и недоработки в документации, подготовленной группой обучения пользователей.

Нельзя совмещать должности тестера и разработчика. Разделение этих обязанностей:

- гарантирует независимую проверку того, что продукт действительно выполняет все требования;
- повышает качество продукта за счет конкуренции между группами.

Хотя проверяют качества продукта только тестеры, за выпуск хорошего продукта отвечают все члены проектной группы.

Контроль изменений

При работе над проектом необходимо контролировать изменения, им должны заниматься все участники группы, но чаще всего в полном объеме этим приходится заниматься именно группе тестирования. Для управления изменениями необходимо:

- создать эталонный документ;
- определить изменяемые элементы;
- определить влияние изменений на существующие системы, •процессы или документы;
- определить метод реализации изменений;
- назначить человека, который внесет изменения;
- определить влияние изменения на условия выполнения проекта, •его бюджет, график и политику;
- получить одобрение изменений (скажем, у руководителя проекта);
- внести изменения;
- сообщать новый документ, в котором изменение учтено.

Прочие обязанности

Некоторые важные обязанности тестеров часто упускают из виду. К ним относятся:

- **уведомление об ошибках и их отслеживание -** тестовая группа отвечает не только за управление изменениями, но и за систему выявления ошибок и информирования о них;
- **сборка продукта -** в группе должен быть человек, ответственный за сборку (компиляцию) продукта, и часто такой «главный сборщик» является тестером, он может использовать только код, хранящийся в системе управления версиями; эту рутинную работе удается автомати-

зировать с помощью сценариев, однако необходимо проверять правильность сборки;

- **выявление и контроль рисков -** это обязанность всех членов группы, менеджер программы должен разработать метод контроля.

Инструктор

Цель группы обучения - повысить эффективность труда пользователей. Поэтому инструкторы «принимают сторону» пользователей подобно тому, как менеджеры продукта представляют интересы заказчика. Однако перед пользователями инструкторы выступают в роли представителей проектной группы.

В этом последнем качестве группа обучения отвечает за выпуск удобного, полезного продукта, которому практически не нужна поддержка. Персонал группы тестирует удобство использования продукт та, выявляет проблемы в этой области и проверяет проект пользовательского интерфейса.

Активно участвуя в создании пользовательского интерфейса, инструкторы сокращают затраты на сопровождение продукта и поддержку пользователей.

Логистик

Логистик представляет интересы служб поддержки и сопровождения, справочных служб и других служб канала доставки. Он занимается развертыванием продукта и его сопровождением и контролирует продукт с этой точки зрения в процессе проектирования. Кроме того, его задача - составление графиков развертывания приложения. Логистики, менеджеры продукта и менеджеры программы совместно определяют порядок передачи продукта пользователям и организации, после чего логистики готовят их к развертыванию приложения.

Размер группы логистики определяется графиком развертывания, уровнем автоматизации установки, наличием средств автоматического развертывания приложений и другими характеристиками этого процесса.

Размеры группы и масштаб проекта

В проектной группе за каждое направление должен отвечать как минимум один человек. При реализации крупного проекта возникает затруднение, связанное с эффективным обменом информацией. В небольших организациях или при работе над мелкими проектами роли можно совмещать.

Крупные проекты

Чтобы справиться с крупным проектом, необходимо делить проектную группу на тематические и функциональные подгруппы.

Тематические группы

Это небольшие подгруппы из одного или нескольких человек, роли которых различны. Каждой из таких групп выделяется некий набор функциональных возможностей приложения, за все стороны проектирования и разработки которого она и отвечает (включая составление проекта и графика реализации). Например, какой-либо группе нужно предоставить решать задачу вывода данных на печать.

В такой группе высока ответственность. Ее члены могут обратиться ко всем людям, опыт которых необходим в их работе. Поэтому, если они так и не найдут оптимального решения, в этом они смогут винить только себя. Такая группа сбалансирована. Вряд ли вы захотите, чтобы окончательные спецификации создавал только отдел разработки, маркетинга или контроля качества. Только решение, принятое группой представителей каждого из этих отделов, будет по-настоящему сбалансировано.

Функциональные группы

Функциональные группы формируются в рамках одной роли. Они нужны в очень крупных проектных группах или при работе над крупномасштабными проектами, когда отдельные роли нуждаются в дополнительном подразделении. Например, в Microsoft отдел менеджмента продукта обычно состоит из групп планирования и маркетинга. Обе они занимаются менеджментом продукта, но первая отвечает за определение действительно необходимых заказчику функций приложения, а вторая - за информирование потенциальных клиентов о достоинствах продукта.

Небольшие проекты

Некоторые должности можно совмещать. Конечно, основной смысл такого разделения в том, чтобы каждую из шести задач решал один из членов группы. Однако, не во всех проектах это возможно.

В небольших группах один человек может играть несколько ролей. При этом мы рекомендуем соблюдать следующие принципы разделения должностей.

- *Нельзя совмещать разработку с другими видами деятельности* - ее создателей приложения не стоит отвлекать от основной задачи. Если «повесить» на разработчиков дополнительные обязанности, то скорее всего график работ будет нарушен, а дату выпуска продукта придется отодвинуть.
- *Конфликт интересов* — нельзя совмещать роли, интересы которых противоположны. Пример - менеджер продукта и менеджер программы. Первый хочет выполнить все требования заказчика, второму же надо уложиться в график и бюджет. Если совместить эти роли, возникает опасность упустить просьбу заказчика о внесении изменений в проект либо, напротив, принять их без должного анализа влияния на график работ. Таким образом, назначение на эти роли разных людей позволяет соблюсти интересы всех участников проекта.

Создание группы

Модель проектной группы описывает структуру группы для работы над проектом создания приложений масштаба предприятия. Однако одной структуры недостаточно - важным фактором успеха является квалификация членов группы.

Поиск руководителей

Главная задача человека, ответственного за создание проектной группы, - подобрать квалифицированных исполнителей. Эта кажущаяся простой (но на самом деле сложная) задача имеет огромное значение для успеха всего проекта.

Найти лидеров - несложная проблема; в любой организации они всем известны. Важно понимать, что говорится именно о лидерах, а не начальниках. Конечно, в любой организации есть менеджеры директора и так далее, но положение в иерархической структуре далеко не всегда гарантирует наличие качеств лидера. Лидеров определяют действия и качества, а не должности.

Руководители должны обладать:

- умением понимать и помогать;
- коммуникабельностью;
- авторитетом внутри организации и за ее пределами;
- чувством ответственности за поставленные цели;
- умением принимать конструктивные решения;
- уверенностью в своих силах;
- достаточной для решения поставленных задач квалификацией;
- способностью помочь другим развить свои таланты и приобрести опыт.

Настоящего лидера отличают именно эти *способности, а* не *намерения.* Это относится и к качествам руководителя.

Повышение эффективности коллективной работы

- заинтересованность;
- надежда, оптимизм, готовность к работе;
- определение задач и решений;
- проявление взаимопомощи;
- доверительные уважительные отношения;
- единение.

Последнее очень важно: эффективность работы группы при этом выше всего.

Для успеха проекта недостаточно только распределить роли и обязанности. Помимо структуры, следует придерживаться определенных принципов и методов

Общее представление о проекте

Важнейший фактор для успеха проекта - единое понимание целей и задач проекта всеми участниками. Каждый из них изначально имеет **свое** мнение, касающееся приложения. В процессе формирования общего представления о проекте все эти мнения обсуждаются, что позволяет добиться единства целей всех членов проектной группы и заказчика.

На основе выработанного представления о проекте создается документ «Концепция проекта», который:

- описывает не только то, что делает продукт, но и то, чего он не делает;
- конкретизирует продукт (например, позволяет включать и исключать определенные функциональные возможности из данной версии); •побуждает группу достичь сформулированной цели;
- содержит описание путей реализации проекта, благодаря чему проектная группа и заказчик могут начать работу.

Задания для практического занятия:

1.Распределение ролей в бригаде (см. приложение).
2.Выполнить работу в соответствии с заданием преподавателя.
3.Оформить отчет

Вопросы для закрепления теоретического материала к практическому занятию:

1. Основные модели организации коллектива при разработке ПО:
2. Недостатки коллективного подхода.
3. Обязанности членов группы.
4. Модель проектной группы. Цели и роли
5. Задачи проектной группы.

Порядок выполнения отчета по практической работе

1. Защита отчета по практической работе заключается в предъявлении преподавателю полученных результатов (на экране монитора и печатном виде), демонстрации полученных навыков и ответах на вопросы преподавателя.
2. Ответить на контрольные вопросы.

ПРИЛОЖЕНИЕ 1

Оформление отчета по практической работе


Образец отчета по практической работе

Дисциплина:

Отчет по практической работе №

Наименование:

Цель:

Ответы на вопросы:

ОБРАЗЕЦ

Выводы:

Выполнил:

Фамилия

Лист

№

Пояснения к оформлению (пример)

Наименование: Коллективная разработка программного продукта

Цель: Приобрести опыт работы в составе бригады при разработке программного продукта

Ответы на вопросы:

1. Основные теоретические положения

Задачами проектной группы являются ….

2.Порядок выполнения работы (таблицы, скриншоты, ход выполнения работы)

1. …

Выводы:

1. …

2. …

Выполнил:

Дата:

Подпись:

ПРИЛОЖЕНИЕ 2

Практические работы выполняются для одного и того же варианта. Преподаватель может изменить содержание варианта по своему усмотрению.

1. Разработать программный модуль «Учет успеваемости студентов». Программный модуль предназначен для оперативного учета успеваемости студентов в сессию. Сведения об успеваемости студентов должны храниться в течение всего срока их обучения и использоваться при составлении справок о прослушанных курсах и приложений к диплому.
2. Разработать программный модуль «Личные дела студентов». Программный модуль предназначен для получения сведений о студентах сотрудниками отдела кадров. Сведения должны храниться в течение всего срока обучения студентов и использоваться при составлении справок и отчетов.
3. Разработать программный модуль «Цикловая комиссия», содержащий сведения о сотрудниках цикловой комиссии (ФИО, должность, ученая степень, дисциплины, нагрузка, общественная работа, совместительство и др.). Модуль предназначен для использования сотрудниками отдела кадров и учебной части.
4. Разработать программный модуль «Лаборатория», содержащий сведения о сотрудниках лаборатории (ФИО, пол, возраст, семейное положение, наличие детей, должность, ученая степень). Модуль предназначен для использования сотрудниками профкома и отдела кадров.
5. Разработать программный модуль «Автосервис». При записи на обслуживание заполняется заявка, в которой указываются ФИО владельца, марка автомобиля, вид работы, дата приема заказа и стоимость ремонта. После выполнения работ распечатывается квитанция.
6. Разработать программный модуль «Учет нарушений правил дорожного движения». Для каждой автомашины (и ее владельца) в базе хранится список нарушений. Для каждого нарушения фиксируется дата, время, вид нарушения и размер штрафа. При оплате всех штрафов машина удаляется из базы.
7. Разработать программный модуль «Картотека агентства недвижимости», предназначенный для использования работниками агентства. В базе содержатся сведения о квартирах (количество комнат, этаж, метраж и др.). При поступлении заявки на обмен (куплю, продажу) производится поиск подходящего варианта. Если такого нет, клиент заносится в клиентскую базу и оповещается, когда вариант появляется.
8. Разработать программный модуль «Картотека абонентов АТС». Картотека содержит сведения о телефонах и их владельцах. Фиксирует задолженности по оплате (абонентской и повременной). Считается, что повременная оплата местных телефонных разговоров уже введена.
9. Разработать программный модуль «Авиакасса», содержащий сведения о наличии свободных мест на авиамаршруты. В базе должны содержаться сведения о номере рейса, экипаже, типе самолета, дате и времени вылета, а также стоимости авиабилетов (разного класса). При поступлении заявки на билеты программа производит поиск подходящего рейса.
10. Разработать программный модуль «Книжный магазин», содержащий сведения о книгах (автор, название, издательство, год издания, цена). Покупатель оформляет заявку на нужные ему книги, если таковых нет, он заносится в базу и оповещается, когда нужные книги поступают в магазин.
11. Разработать программный модуль «Автостоянка». В программе содержится информация о марке автомобиля, его владельце, дате и времени въезда, стоимости стоянки, скидках, задолженности по оплате и др.
12. Разработать программный модуль «Кадровое агентство», содержащий сведения о вакансиях и резюме. Программный модуль предназначен как для поиска сотрудника, отвечающего требованиям руководителей фирмы, так и для поиска подходящей работы.

Варианты предметной области для выполнения практических работ

1. Приемная комиссия вуза (абитуриенты, экзаменаторы, предметы, оценки; справочные сведения о подразделениях учебного заведения).
2. Успеваемость студентов (зачеты, экзамены, преподаватели, предметы; результаты сессии, перевод на следующий курс, отчисление).
3. Учебный план (преподаватели, предметы, виды занятий, плановая и фактическая нагрузка, категории преподавателей).

4. Расписание занятий (дни, часы, аудитории, предметы, преподаватели, учебные группы; ограничения для студентов и преподавателей).

5. Учет выполнения лабораторных работ (темы работ, предметы, преподаватели; план выполнения работ, исполнение плана, ограничения на выполнение работ).

6. Аспиранты кафедры (аспиранты, руководители, специальности, темы сроки и форма обучения, аттестация, выпуск, конференции).

7. Кадровый учет предприятия (штатное расписание, зарплата, отделы предприятия, заполнение потребность в специалистах, требования к специалистам).

8. Выполнение заказов на изготовление изделий (заказчики, исполнители, материалы, изделия, поставщики).

9. Предприятие по сборке, комплектации и продаже персональных компьютеров и периферийного оборудования (клиенты, заказы, поставщики, комплектующие, программные средства, сотрудник)

10. Ремонтная мастерская (клиент, заказ, изделие, комплектующие, категория клиентов, исполнитель).

11. Организация работы интернет-кафе (программное обеспечение, оборудование, оплата и предоставление услуг, персонал, клиенты).

12. Гостиница (список номеров и их категории, занятость, сроки заезда и отъезда, продление, оплата, клиенты и персонал).

13. Туристическая фирма (путевки, туроператоры, клиенты - организации и физические лица, лимит путевок, скидки, категории клиентов).

14. Агентство недвижимости (квартиры, договор, оплата услуг, клиенты, персонал).

15. Служба доставки (клиенты, график доставки, транспорт, маршрут, исполнитель).

16. Железная дорога (поезд, пассажир, билет, класс, услуги, ограничения).

17. Видеопрокат (фильмы, клиенты, категории клиентов, служащие, категории фильмов)

18. Магазин заказов (заказчики, заказы, закупки, выдача и оплата заказов, отчетность).

19. Аптека (покупатели, лекарства, заменители лекарства, склад, поставщики, служащие)

20. Учет товаров на складе (товар, категория, материально ответственные лица, накладная, поставщик).

21. Интернет-провайдер (трафик, пользователь, тарифные планы, скидки).

22. Банковские услуги (Клиент, счет, виды вкладов, операция, кредиты, исполнитель).

23. Библиотека вуза (получение и регистрация книг, формирование каталога по тематике, выдача книг, списание; учет читателей).

24. Каталог компакт-дисков (поступление и списание дисков, типы и справки в зависимости от типов, выдача, возврат, копирование).

25. Земельный кадастр (расположение участков, их качество, стоимость, форма собственности, владелец, рейтинг).

26. Учет жилищного фонда (улицы, дома, квартиры, их состояние, населенность, и т.п.).

Примечание. При разработке программы не ограничиваться функциями, приведенными в варианте, добавить несколько своих функций. Обязательно использование структурного и модульного подходов к программированию. Желательно использование объектного подхода.

ПРИЛОЖЕНИЕ 3

Пример разработки технического задания на программный продукт

1.Титульный лист

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ САМАРСКОЙ ОБЛАСТИ

ГБОУ СПО «Тольяттинский машиностроительный колледж»

Техническое задание

на разработку «Модуля автоматизированной системы
оперативно-диспетчерского управления теплоснабжением корпусов
Тольяттинского машиностроительного колледжа»

Руководитель преподаватель _____ Громова Л.Н.

Исполнитель студент гр. ВТ 25-1 _____Иванов А.А.

Тольятти, 2012 г.

2. Пояснительная записка

1. Введение

Работа выполняется в рамках проекта «Автоматизированная система оперативно-диспетчерского управления теплоснабжением корпусов Тольяттинского машиностроительного колледжа».

2. Основание для разработки

2.1 Основанием для данной работы служит договор № 1234 от 10 сентября 2010 г.

2.2 Наименование работы:

«Модуль автоматизированной системы оперативно-диспетчерского управления теплоснабжением корпусов Тольяттинского машиностроительного колледжа».

2.3 Исполнители: ОАО «Лаборатория создания программного обеспечения».

2.4 Соисполнители: нет.

3.Назначение разработки

Создание модуля для контроля и оперативной корректировки состояния основных параметров теплообеспечения корпусов Тольяттинского машиностроительного колледжа.

4.Технические требования

4.1. Требования к функциональным характеристикам.

4.1.1. Состав выполняемых функций.

Разрабатываемое ПО должно обеспечивать:

- сбор и анализ информации о расходовании тепла, горячей и холодной воды по данным теплосчетчиков SA-94 на всех тепловых выходах;
- сбор и анализ информации с устройств управления системами воздушного отопления и кондиционирования типа РТ1 и РТ2 (разработки лабораторий СММЭ и ТЦ);
- предварительный анализ информации на предмет нахождения параметров в допустимых пределах и сигнализирование при выходе параметров за пределы допуска;
- выдачу рекомендаций по дальнейшей работе;
- отображение текущего состояния по набору параметров –циклически/постоянно (режим работы круглосуточный), при сохранении периодичности контроля прочих параметров;
- визуализацию информации по расходу теплоносителя:

а) текущую, аналогично показаниям счетчиков;

б) с накоплением за прошедшие сутки, неделю, месяц - в виде почасового графика для информации за сутки и неделю;

в) суточный расход - для информации за месяц.

Для устройств управления приточной вентиляцией текущая информация должна содержать номер приточной системы и все параметры, выдаваемые на собственный индикатор.

По отдельному запросу осуществляются внутренние настройки.

В конце отчетного периода система должна архивировать данные.

4.1.2. Организация входных и выходных данных.

Исходные данные в систему поступают в виде значений с датчиков, установленных в помещениях колледжа. Эти значения отображаются на компьютере диспетчера. После анализа поступившей информации оператор диспетчерского пункта устанавливает необходимые параметры для устройств, регулирующих отопление и вентиляцию в помещениях. Возможна также автоматическая установка некоторых параметров для устройств регулирования.

Основной режим использования системы - ежедневная работа.

4.2. Требования к надежности.

Для обеспечения надежности необходимо проверять корректность получаемых данных с датчиков.

4.3. Условия эксплуатации и требования к составу и параметрам технических средств.

Для работы системы должен быть выделен ответственный оператор.

Требования к составу и параметрам технических средств уточняются на этапе эскизного проектирования системы.

4.4. Требования к информационной и программной совместимости.

Программа должна работать на платформах Windows XP/ Windows-7.

4.5. Требования к транспортировке и хранению.

Программа поставляется на лазерном носителе информации. Программная документация поставляется в электронном и печатном виде.

4.6. Специальные требования:

- программное обеспечение должно иметь дружественный интерфейс, рассчитанный на пользователя (в плане компьютерной грамотности) квалификации;
- ввиду объемности проекта задачи предполагается решать поэтапно, при этом модули ПО, созданные в разное время, должны предполагать возможность наращивания системы и быть совместимы друг с другом, поэтому документация на принятое эксплуатационное ПО должна содержать полную информацию, необходимую для работы программистов с ним;
- язык программирования - по выбору исполнителя, должен обеспечивать возможность интеграции программного обеспечения с некоторыми видами периферийного оборудования (например, счетчик SA-94 и т. п.).

5. Требования к программной документации

Основными документами, регламентирующими разработку будущих программ, должны быть документы Единой Системы Программной Документации (ЕСПД): руководство пользователя, руководство администратора, описание применения.

6. Технико-экономические показатели

Эффективность системы определяется удобством использования системы для контроля и управления основными параметрами теплообеспечения помещений Тольяттинского машиностроительного колледжа, а также экономической выгодой, полученной от внедрения аппаратно-программного комплекса.

7. Порядок контроля и приемки

После передачи Исполнителем отдельного функционального модуля программы Заказчику последний имеет право тестировать модуль в течение 7 дней. После тестирования Заказчик должен принять работу по данному этапу или в письменном виде изложить причину отказа принятия. В случае обоснованного отказа Исполнитель обязуется доработать модуль.

8. Календарный план работ

№ этапа	Название этапа	Сроки этапа	Чем заканчивается этап
1	Изучение предметной области. Проектирование системы. Разработка приложений по реализации системы.	01.02.20__ - 28.02.20___	Предложения по работе системы. Акт сдачи-приемки
2	Разработка программного модуля по сбору и анализу информации со счетчиков и устройств управления. Внедрение системы для одного из корпусов	01.03.20__ - 31.08.20___	Программный комплекс, решающий поставленные задачи для корпуса «А». Акт сдачи-приемки.
3	Тестирование и отладка модуля. Внедрение системы во всех корпусах ТМК	01.09.20__ - 30.12.20___	Готовая система контроля теплообеспечения ТМК, установленная в диспетчерском пункте. Программная документация. Акт сдачи-приемки работ.

Руководитель работ_____Сидоров С. В.

ПРИЛОЖЕНИЕ 4

Пример оформления эскизного проекта

1. Титульный лист

УТВЕРЖДАЮ
Руководитель (заказчика ИС)
Личная подпись _____ Расшифровка подписи _____
Печать
Дата «_____» _____ 20____ г.

УТВЕРЖДАЮ
Руководитель (разработчика ИС)
Личная подпись _____ Расшифровка подписи _____
Печать
Дата «_____» _____ 20____ г.

Эскизный проект на создание информационной системы

Система управления базой данных
(наименование вида ИС)

Пенсионный фонд
(наименование объекта информатизации)

СУБД «Пенсионный фонд»
(сокращенное наименование ИС)

На ____ листах

Действует с «____» _____ 20____ г.

2. Пояснительная записка

Содержание

Ведомость эскизного проекта

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

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

Пояснительная записка к эскизному проекту

Общие положения

Данный документ является эскизным проектом на создание Системы Управления Базой Данных для Пенсионного Фонда Автозаводского района г. Тольятти (СУБД «Пенсионный фонд»).

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

Основные технические решения

Решения по структуре системы

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

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

Общая структура базы данных:

- **Анкеты организации, которые зарегистрированы в данном ПФ.**

1. Тип предприятия (Российская организация, Физическое лицо, Иностранная организация, Обособленное подразделение).
2. Вид предприятия (Адвокаты, Бюджетное, Единый налог 6%, Единый налог 15%, Сельхозпродукция, Службы занятости, Фермерское хозяйство, Прочее).
3. Регистрационный номер работодателя в ПФР (3 - 3 - 6).
4. Свидетельство: серия, номер.
5. Дата выдачи свидетельства (число, месяц, год).
6. ИНН.
7. КПП.
8. Наименование.
9. Юридический адрес:

- Почтовый индекс.
- Регион.
- Район.
- Город.
- Населенный пункт.
- Улица.

- Дом.
- Корпус.
- Квартира.
- Адрес постоянно действующего органа (при отличии от юридического).

- **Анкеты сотрудников этих организаций.**

1. Фамилия.
2. Имя.
3. Отчество.
4. Пол (М/Ж).
5. Дата рождения (Дата).
6. Страховой номер.
7. Место рождения (Страна, Регион, Район, Город, Населенный пункт).
8. Гражданство.
9. Адрес регистрации (Страна, Почтовый индекс, Регион, Район, Город, Населенный пункт, Улица, Дом, Корпус, Квартира).
10. Адрес места жительства фактический (Страна, Почтовый индекс, Регион, Район, Город, Населенный пункт, Улица, Дом, Корпус, Квартира).
11. Телефон домашний.
12. Телефон служебный.
13. Документ (Удостовер. личность).
14. Дата выдачи (Дата).
15. Кем выдан ().
16. Дата заполнения (Дата).
17. ИНН.

- **Сведения о стаже сотрудников этих организаций.**

1. Страховой номер.
2. Фамилия.
3. Имя. Отчество.
4. Дата рождения.
5. Территориальные условия проживания на ....
6. Таблица периодов работы со следующей структурой:

- Начало периода (дата).
- Конец периода (дата).
- Вид деятельности (работа, служба соцстрах, уход-дети, безр, реабилит, уход-инвд, профзаб, пересмотр).
- Наименование организации.
- Должность.
- Территориальные условия.

Решения по режимам функционирования, работы системы

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

- просматривать записи базы данных (в том числе и при помощи фильтров);
- добавлять новые записи;
- удалять записи;
- при входе в систему будет запрашиваться пароль.

Решения по численности, квалификации и функциям персонала АС

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

Состав функций комплексов задач, реализуемых системой

Автоматизированная система должна выполнять следующие функции:

- сделать запись о пенсионном удостоверении;
- удалить информацию о пенсионном удостоверении;
- выдать справку обо всех пенсионных удостоверениях;
- зарегистрировать новое предприятие в ПФ РФ;
- удалить предприятие из базы данных;
- выдать справку обо всех предприятиях, зарегистрированных в ПФ РФ;

- подсчитать пенсию для работников предприятий на основании стажа;
- выдать справку о пенсионных накоплениях работника.

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

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

Для подсчета пенсии будет использоваться следующий алгоритм.

1. Вначале определяется стажевый коэффициент пенсионера. Он полагается равным 0,55 за общий трудовой стаж до текущей даты не менее 25 лет мужчинам и 20 лет женщинам. За каждый полный год стажа сверх указанного стажевый коэффициент увеличивается на 0,01, но не более чем на 0,20.
2. Затем определяется отношение заработка пенсионера к среднемесячной заработной плате в стране. Этот заработок может быть взят за этот отсчетный период или за любые 60 месяцев работы подряд, или тот, из которого была исчислена пенсия на момент реформы. Среднемесячная зарплата в стране берется за тот же самый период.
3. Отношение заработков учитывается в размере не свыше 1,2. Для пенсионеров, проживающих на Крайнем Севере, учитываемое соотношение выше: от 1,4 до 1,9 в зависимости от установленного в централизованном порядке районного коэффициент к зарплате.
4. Затем стажевый коэффициент умножается на соотношение заработков и на 1671 руб. - утвержденную для расчетов среднемесячную зарплату в стране за III квартал 2001 г. Это и будет пересчитанный размер трудовой пенсии по новому законодательству в обычном случае. Если он оказался менее 660 руб., то размер пенсии «доводится» до этого гарантированного минимума
5. Если пенсионер является инвалидом I группы или достиг к 1 января 2010 г. возраста 80 лет и более, рассчитанный в этом порядке размер пенсии по старости увеличивается на 450 руб.
6. Если у пенсионера имеются лица, находящиеся на его иждивении, то рассчитанный размер пенсии увеличивается на 150 руб. на каждого иждивенца, но не более чем на трех в общей сложности.

Источники разработки

Данный документ разрабатывался на основании ГОСТ 34.698-90 на написание ТЗ на автоматизированные системы управления от 01.01.2010 г.

Приложения

СОСТАВИЛИ
Должность исполнителя _____
Фамилия, имя, отчество _____
Подпись _____
Дата «_____» _____ 2010 г.

Должность исполнителя _____
Фамилия, имя, отчество _____
Подпись _____
Дата «_____» _____ 2010 г.

Должность исполнителя _____
Фамилия, имя, отчество _____
Подпись _____
Дата «_____» _____ 2010 г.

ПРИЛОЖЕНИЕ 5

ДИАГРАММЫ ПРЕЦЕДЕНТОВ

В соответствии с методологией объектно-ориентированного анализа и проектирования первым этапом является анализ требований, который подразумевает выделение процессов и требований и их формулировку в виде прецедентов.

Прецедент в объектном моделировании (англ. – use case) представляет собой документ, описывающий последовательность событий, связанных с исполнителем (внешним агентом), который для завершения требуемого процесса использует создаваемую систему. Прецеденты являются описанием или вариантами использования системы. С помощью прецедента описывается некоторый процесс.

По результатам анализа прецедентов на первом этапе моделирования предметной области создается диаграмма определения требований к системе Use Case (сценарии поведения). Данная диаграмма позволяет создавать диаграммы поведения объектов системы.

На диаграмме прецедентов (рисунок 1) иллюстрируется набор прецедентов системы и исполнители, а также взаимосвязи между ними. Прецеденты определяют, как исполнители взаимодействуют с программной системой. В процессе этого взаимодействия исполнителем генерируются события, передаваемые системе, которые представляют собой запросы на выполнение некоторой операции.

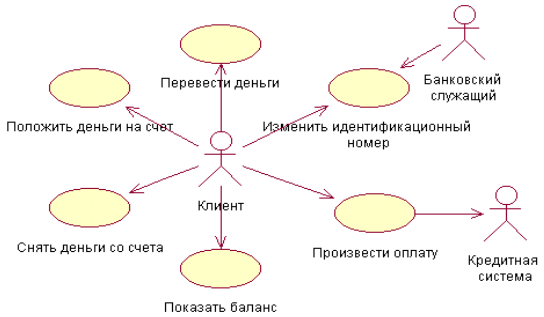


Рисунок 1 – Диаграмма прецедентов, описывающая процесс обслуживания клиента в банке

Диаграмма прецедентов содержит:

- варианты использования (прецеденты) системы (use case);
- действующее лицо (actor).

Диаграмма отражает взаимодействие вариантов использования и действующих лиц. Она отражает требования к системе с точки зрения пользователя.

Варианты использования системы – описание функций системы на «высоком уровне». Они описывают все, что происходит внутри области действия системы. Варианты использования иллюстрируют, как можно использовать систему. Они заостряют внимание на том, что пользователи хотят получить от системы. Каждый вариант использования представляет собой завершенную транзакцию между пользователем и системой.

Действующее лицо – все, что взаимодействует с системой, передает или получает информацию от системы. Исполнитель (actor) является внешним по отношению к системе понятием, которое определенным образом участвует в процессе, описываемом прецедентом. Они описывают все, что находится вне системы. Это пользователи системы, другие системы, взаимодействующие с описываемой, время.

Каждый прецедент должен быть инициирован действующим лицом.

Как правило, отдельные шаги или виды деятельности в виде прецедента не представляются.

Часто для одной системы создается несколько диаграмм Вариантов Использования. На диаграмме высокого уровня (Main) указываются только пакеты вариантов использования. Другие диаграммы описывают совокупности вариантов использования и действующих лиц.

Цель диаграмм – документирование вариантов использования, действующих лиц и связей между ними. Разрабатывая диаграммы, придерживаются правил:

1. Не моделируют связи между действующими лицами. По определению они находятся вне сферы действия системы. Связи между ними не относятся к ее компетенции.
2. Не соединяют стрелкой непосредственно два варианта использования (кроме связей использования и расширения). Диаграмма описывает только, какие варианты использования доступны системе, а не порядок их выполнения.
3. Каждый вариант использования должен быть инициирован действующим лицом. Всегда должна быть стрелка, начинающаяся на действующем лице и заканчивающаяся на варианте использования (кроме связей использования и расширения).
4. Думают о БД, как о слое, находящемся под диаграммой. С помощью одного варианта использования можно вводить данные в базу, а получать их – с помощью другого. Не рисуют стрелки от одного варианта к другому для изображения потока информации.

При создании диаграмм прецедентов вначале определяются исполнители (роли, пользователи) (рисунок 2).

Исполнитель может быть абстрактным, не имеющим экземпляров. Например, есть несколько действующих лиц: служащий с почасовой оплатой, служащий с окладом и т.д. Все они являются разновидностями действующего лица - служащего. Абстрактный исполнитель существует для того, чтобы показать общность между этими типами.

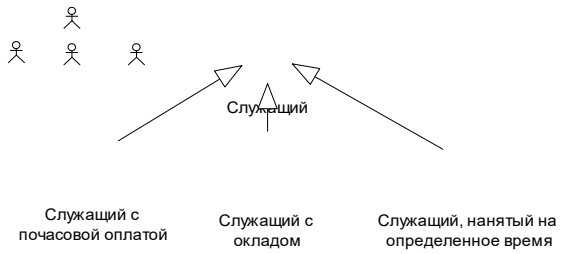


Рисунок 2 - Определение ролей исполнителей

Следующий шаг - идентификация прецедентов.

У каждого прецедента должно быть уникальное имя.

Каждый прецедент должен иметь связанное с ним короткое описание того, что он будет делать. Следует делать описание коротким и к «месту», при этом оно должно определять типы пользователей, выполняющих вариант использования, и ожидаемый ими конечный результат.

Предусловия варианта использования – это такие условия, которые должны быть выполнены, прежде чем вариант начнет вою работу. Например: это может быть выполнение другого варианта использования или наличие у пользователя прав доступа, требуемых для запуска данного варианта использования.

Постусловия – такие условия, которые должны быть выполнены после завершения варианта использования. С помощью постусловий можно вводить сведения о порядке выполнения вариантов использования системы. Если, скажем, после одного варианта использования должен всегда выполняться другой, это можно описать как постусловие. Такие условия имеются не у каждого варианта использования.

Конкретные детали вариантов использования отражаются в основном и альтернативном потоках событий. Поток событий поэтапно описывает, что должно происходить во время выполнения заложенной в варианты использования функциональности. Поток событий уделяет внимание тому, что (а не как) будет делать система, причем описывает это с точки зрения пользователя. Первичный и альтернативный потоки событий содержат:

- описание того, каким образом запускается вариант использования,
- различные пути выполнения варианта использования,
- нормальный, или основной, поток событий варианта использования,
- отклонения от основного потока событий (так называемые альтернативные потоки),
- потоки ошибок,
- описание того, каким образом завершается вариант использования.

Поток событий должен быть согласован с определенными ранее требованиями. Избегают детальных обсуждений того, как поток будет реализован.

Можно создать подробную спецификацию для каждого варианта использования. Они помогают документировать такие атрибуты вариантов использования, как имена, приоритеты и стереотипы.

В диаграммах прецедентов поддерживается несколько типов связей:

- связи коммуникации (Communication) – описывают связи между действующими лицами и вариантами использования;
- использования (uses) и расширения (extends) – отражают связи между вариантами использования;
- обобщения действующего лица (actor generalization) – между действующими лицами.

Связь использования позволяет одному варианту использования задействовать функциональность другого. С помощью таких связей обычно моделируют многократно применяемую функциональность, встречающуюся в двух или более вариантах использования.

Связь расширения позволяет варианту использования только при необходимости применять функциональные возможности другого варианта (extends).

На основе набора Use Case диаграмм создается список требований к системе и определяется множество выполняемых системой функций.

ДИАГРАММЫ ВЗАИМОДЕЙСТВИЯ

Диаграммы взаимодействия отображают один из процессов обработки информации в рамках ***варианта использования***. В варианте использования может быть несколько альтернативных потоков. Это значит, что для данного варианта использования нужно создать несколько диаграмм взаимодействия, отражающих один и тот же процесс в различных условиях (одна показывает, что происходит, когда все в порядке, другая, что произойдет в случае ошибки и т.д.).

Диаграммы взаимодействия делятся на диаграммы последовательности (Sequence diagram) и кооперативные диаграммы (Collaboration diagram).

На диаграммах обоих типов может быть представлена одна и та же информация, однако диаграммы последовательности заостряют внимание на управлении, а кооперативные отображают потоки данных.

Диаграммы последовательности упорядочены во времени. Они полезны для того, чтобы понять логическую последовательность событий. Кооперативные диаграммы показывают, как компоненты системы взаимодействуют друг с другом.

Диаграммы взаимодействия содержат «Объекты» и «Сообщения». С помощью сообщения один объект или класс запрашивает у другого выполнения какой-то конкретной функции, например, форма может запросить у объекта отчет напечатать ее.

Главное здесь: объекты, которые должны быть созданы для реализации функциональных возможностей, заложенных в вариант использования. На диаграммах последовательности и кооперативных диаграммах могут быть показаны объекты, классы или то и другое вместе.

Объект – это абстракция чего-либо в домене прикладной области или в выполняемой системе. Например, объектом может быть счет в бизнес системе или служащий в системе платежной ведомости.

Объект инкапсулирует данные и поведение, которые отличаются от традиционного разделения на функции и данные. Данные объекта представляются атрибутами, а его поведение - операциями. Значения атрибутов изменяются время от времени, но сами атрибуты неизменны.

Класс — это некая сущность, представляющая собой как бы схему объекта. Иными словами, класс определяет данные и поведение, которыми должен обладать объект. Класс — более общий термин, являющийся, по существу, шаблоном для объектов.

Сообщение (message) - это связь между объектами, в которой один из них (клиент) требует от другого (сервера) выполнения каких-то действий. При генерации кода сообщения транслируются в вызовы функций.

На диаграмме последовательности взаимодействие изображается в виде двухмерной схемы (в формате графа или сети). По вертикали проходит временная ось, где течение времени происходит сверху вниз. По горизонтали указываются роли классификатора, которые представляют отдельные объекты кооперации. У каждой роли классификатора есть «линия жизни», идущая сверху вниз. Тот период времени, в течение которого объект существует, изображается на диаграмме вертикальной пунктирной линией. Во время вызова процедуры определенного объекта (активизации) его линия жизни изображается двойной линией.

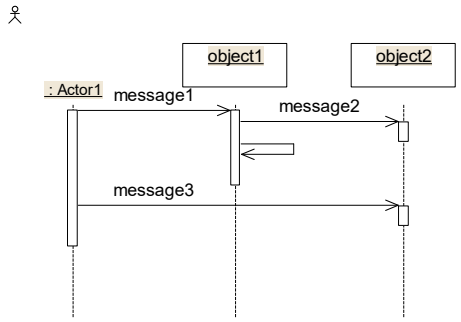


Рисунок 3 - Диаграмма последовательности

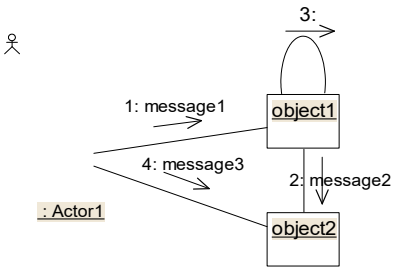


Рисунок 4 - Кооперативная диаграмма

Активацией называется выполнение процедуры, включающее в себя время ожидания выполнения всех вложенных процедур. На диаграмме активация изображается в виде двойной линии. Которая замещает собой часть линии жизни объекта. Активным называется объект, которому принадлежит стек активаций (и вызовов операций). У каждого активного объекта есть свой собственный поток управления, который выполняется параллельно с другими активными объектами. Объекты, вызываемые пассивными объектами, называется пассивными. Они получают управление только на время вызова, а потом возвращают его.

Диаграмма кооперации – это диаграмма классов, на которой отображаются не просто классификаторы и ассоциации, а роли классификатора и роли в ассоциации. Роли классификатора и роли в ассоциации описывают конфигурацию объектов и связей, которые могут образоваться при выполнении кооперации в реальной системе.

С помощью диаграмм Взаимодействия проектировщики и разработчики системы могут определить классы, которые нужно создать, связи между ними, а также операции и ответственности (responsibilities) каждого класса. Диаграммы Взаимодействия - краеугольный камень, на котором возводится оставшаяся часть проекта.

ДИАГРАММЫ ДЕЙСТВИЙ

При моделировании поведения проектируемой системы возникает необходимость не только представить процесс изменения её состояний, но и детализировать особенности алгоритмической и логической операции выполняемой системой реализации. Традиционно для этой цели использовались блок-схема или структурные схемы алгоритмов. В UML для этого используется диаграмма действий.

На диаграмме деятельности отображается логика или последовательность переходов от одной деятельности к другой. При этом внимание фиксируется на результате деятельности. Сам же результат может привести к изменению состояния системы или возвращению некоторого значения.

Графически диаграмма деятельности представляется в форме графа деятельности, вершинами которого является состояние действия, а дугами – переходы от одного состояния в действия к другому.

При этом каждое состояние может являться выполнением операции некоторого класса либо её часть, позволяя использовать диаграмму деятельности для описания реакции на внутренние события системы.

На рисунке 5 представлен алгоритм процесса принятия решения при настройке и внедрении программного продукта, описанный с помощью диаграммы действий.

В языке UML действие изображается в виде прямоугольника с закругленными углами, состояния - в виде прямоугольника, переходы – в виде направленных стрелок, элементы выбора – в виде ромбов, линии синхронизации – в виде толстых горизонтальных или вертикальных линий.

Состоянием называется одно из возможных условий, в которых может существовать объект. Для выявления состояний объекта необходимо исследовать две области модели: значения атрибутов объекта и связи с другими объектами. В рассматриваемом примере: «Сформированы требования к ПП», «Выданы рекомендации».

Существуют специальные состояния объекта: начальное и конечное. Начальным (Start State) называется состояние, в котором находится объект сразу после своего создания (В примере это «заказ на ПО»). Конечным (End State) называется состояние, в котором объект находится непосредственно перед уничтожением («Решение по адаптации»).

Действием называется исполнение определенного поведения в потоке управления системы. В примере это «Поиск правила», «Формировать решение» и т.д.

Переходы используются для изображения пути потока управления от действия к действию.

Элементы выбора показывают места разделения управляющих потоков на основе условного выбора. Переходы из элементов выбора содержат ограничительные условия, определяющие, какое направление перехода будет выбрано («Поиск правила», «Поиск прецедента»).

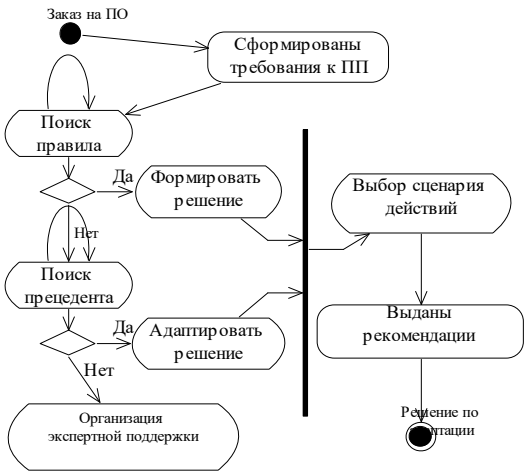


Рисунок 5 - Диаграмма действий

Линии синхронизации (synchronization bar) позволяет указать на необходимость одновременного выполнения действий, а также обеспечивает единое выполнение действий в потоке («Формировать решение», «Адаптировать решение»).

Секции делят диаграмму действий на несколько участков, чтобы показать, кто отвечает за выполнение действий на каждом участке.

Диаграммы действий отражают динамику проекта и представляют собой схемы потоков управления в системе от действия к действию, а также параллельные действия и альтернативные потоки.

CRC-КАРТЫ

Для выделения или идентификации компонентов предметной области было предложено несколько способов и правил. Сам этот процесс получил название ***концептуализации предметной области***. При этом под компонентой понимают некоторую абстрактную единицу, которая обладает функциональностью, т.е. может выполнять определенные действия, связанные с решением поставленных задач. На предварительном этапе концептуализации рекомендуется использовать так называемые CRC-карточки (Component, Responsibility, Collaborator - компонента, обязанность, сотрудники). Для каждой выделенной компоненты предметной области разрабатывается собственная CRC-карточка (рисунок 6).

Компонента (название)	**Список** названий других компонент, с которыми связан данная компонента
Описание обязанностей, выполняемых данной компонентой	

Рисунок 6 - Общий вид CRC-карточки для описания компонентов предметной области

Появление методологии ООАП потребовало, с одной стороны, разработки различных средств концептуализации предметной области, а с другой — соответствующих специалистов, которые владели бы этой методологией. На данном этапе появляется относительно новый тип специалиста, который получил название аналитика или архитектора. Наряду со специалистами по предметной области аналитик участвует в построении концептуальной схемы будущей программы, которая затем преобразуется программистами в код. При этом отдельные компоненты выбираются таким образом, чтобы при последующей разработке их было удобно представить в форме классов и объектов. В этом случае немаловажное значение приобретает и сам язык представления информации о концептуальной схеме предметной области.

ДИАГРАММЫ «СУЩНОСТЬ-СВЯЗЬ»

Данная нотация была предложена П. Ченом (P. Chen) в 1976 году и получила дальнейшее развитие в работах Р. Баркера. Диаграммы «сущность-связь» (Entity-Relationship Diagrams, ERD) предназначены для графического представления моделей данных разрабатываемой программной системы и предлагают некоторый набор стандартных обозначений для определения данных и отношений между ними. С помощью этого вида диаграмм можно описать отдельные компоненты концептуальной модели данных и совокупность взаимосвязей между ними, имеющих важное значение для разрабатываемой системы.

Основными понятиями данной нотации являются понятия сущности и связи. При этом под сущностью (entity) понимается произвольное множество реальных или абстрактных объектов, каждый из которых обладает одинаковыми свойствами и характеристиками. В этом случае каждый рассматриваемый объект может являться экземпляром одной и только одной сущности, должен иметь уникальное имя или идентификатор, а также отличаться от других экземпляров данной сущности.

Примерами сущностей могут быть: банк, клиент банка, счет клиента, аэропорт, пассажир, рейс, компьютер, терминал, автомобиль, водитель. Каждая из сущностей может рассматриваться с различной степенью детализации и на различном уровне абстракции, что определяется конкретной постановкой задачи. Для графического представления сущностей используются специальные обозначения (рисунок 7).

Рисунок 7 - Графические изображения для обозначения сущностей

Связь (relationship) определяется как отношение или некоторая ассоциация между отдельными сущностями. Примерами связей могут являться родственные отношения типа «отец-сын» или производственные отношения типа «начальник-подчиненный». Другой тип связей задается отношениями «иметь в собственности» или «обладать свойством». Различные типы связей графически изображаются в форме ромба с соответствующим именем данной связи (рисунок 8).

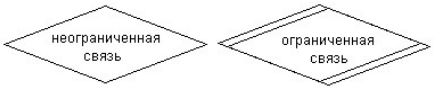


Рисунок 8 - Графические изображения для обозначения связей

Графическая модель данных строится таким образом, чтобы связи между отдельными сущностями отражали не только семантический характер соответствующего отношения, но и дополнительные аспекты обязательности связей, а также кратность участвующих в данных отношениях экземпляров сущностей.

Рассмотрим в качестве простого примера ситуацию, которая описывается двумя сущностями: «Сотрудник» и «Компания». При этом в качестве связи естественно. использовать отношение принадлежности сотрудника данной компании. Если учесть соображения о том, что в компании работают несколько сотрудников, и эти сотрудники не могут быть работниками других компаний, то данная информация может быть представлена графически в виде следу-

ющей диаграммы «сущность-связь» (рисунок 9). На данном рисунке буква «N» около связи означает тот факт, что в компании могут работать более одного сотрудника, при этом значение N заранее не фиксируется. Цифра «1» на другом конце связи означает, что сотрудник может работать только в одной конкретной компании, т. е. не допускается прием на работу сотрудников по совместительству из других компаний или учреждений.

Рисунок 9 - Диаграмма «сущность-связь» для примера сотрудников некоторой компании

Несколько иная ситуация складывается в случае рассмотрения сущностей «сотрудник» и «проект», и связи «участвует в работе над проектом» (рисунок 10). Поскольку в общем случае один сотрудник может участвовать в разработке нескольких проектов, а в разработке одного проекта могут принимать участие несколько сотрудников, то данная связь является многозначной. Данный факт специально отражается на диаграмме указанием букв «N» и «M» около соответствующих сущностей, при этом выбор конкретных букв не является принципиальным.

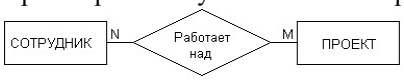


Рисунок 10 - Диаграмма «сущность-связь» для примера сотрудников, участвующих в работе над проектами

Рассмотренные две диаграммы могут быть объединены в одну, на которой будет представлена информация о сотрудниках компании, участвующих в разработке проектов данной компании (рисунок 11). При этом может быть введена дополнительная связь, характеризующая проекты данной компании.

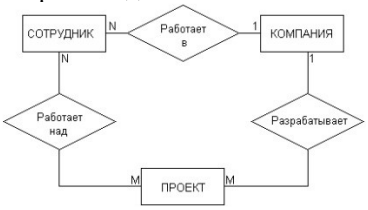


Рисунок 11 - Диаграмма «сущность-связь» для общего примера компании

Примечание

На указанных диаграммах могут быть отражены более сложные зависимости между отдельными сущностями, которые отражают обязательность выполнения некоторых дополнительных условий, определяемых спецификой решаемой задачи и моделируемой предметной области. В частности, могут быть отражены связи подчинения одной сущности другой или введения ограничений на действие отдельных связей. В подобных случаях используются дополнительные графические обозначения, отражающие особенности соответствующей семантики.

Ограниченность ERD проявляется при конкретизации концептуальной модели в более детальное представление моделируемой программной системы, которое кроме статических связей должно содержать информацию о поведении или функционировании отдельных ее компонентов.

ДИАГРАММЫ ФУНКЦИОНАЛЬНОГО МОДЕЛИРОВАНИЯ

Начало разработки диаграмм функционального моделирования относится к середине 1960-х годов, когда Дуглас Т. Росс предложил специальную технику моделирования, получившую название SADT (Structured Analysis & Design Technique). Военно-воздушные силы США использовали методику SADT в качестве части своей программы интеграции компьютерных и промышленных технологий (Integrated Computer Aided Manufacturing, ICAM) и назвали ее IDEFO (Icam DEFinition). Целью программы ICAM было увеличение эффективности компьютерных технологий в сфере проектирования новых средств вооружений и ведения боевых действий. Одним из результатов этих исследований являлся вывод о том, что описательные языки не эффективны для документирования и моделирования процессов функционирования сложных систем. Подобные описания на естественном языке не обеспечивают требуемого уровня непротиворечивости и полноты, имеющих доминирующее значение при решении задач моделирования.

В рамках программы ICAM было разработано несколько графических языков моделирования, которые получили следующие названия:

- Нотация IDEF0 — для документирования процессов производства и отображения информации об использовании ресурсов на каждом из этапов проектирования систем.
- Нотация IDEF1 — для документирования информации о производственном окружении систем.
- Нотация IDEF2 — для документирования поведения системы во времени.
- Нотация IDEF3 — специально для моделирования бизнес-процессов.

Нотация IDEF2 никогда не была полностью реализована. Нотация IDEF1 в 1985 году была расширена и переименована в IDEF1X. Методология IDEF-SADT, нашла применение в правительственных и коммерческих организа-

циях, поскольку на тот период времени вполне удовлетворяла различным требованиям, предъявляемым к моделированию широкого класса систем.

В начале 1990 года специально образованная группа пользователей IDEF (IDEF Users Group), в сотрудничестве с Национальным институтом по стандартизации и технологии США (National Institutes for Standards and Technology, NIST), предприняла попытку создания стандарта для IDEFO и IDEF1X. Эта попытка оказалась успешной и завершилась принятием в 1993 году стандарта правительства США, известного как FIPS для данных двух технологий IDEFO и IDEF1X. В течение последующих лет этот стандарт продолжал активно развиваться и послужил основой для реализации в некоторых первых CASE-средствах.

Методология IDEF-SADT представляет собой совокупность методов, правил и процедур, предназначенных для построения функциональной модели сиетемы какой-либо предметной области. Функциональная модель SADT отображает структуру процессов функционирования системы и ее отдельных подсистем, т. е. выполняемые ими действия и связи между этими действиями. Для этой цели строятся специальные модели, которые позволяют в наглядной форме представить последовательность определенных действий. Исходными строительными блоками любой модели IDEFO процесса являются деятельность (activity) и стрелки (arrows).

Рассмотрим кратко эти основные понятия методологии IDEF-SADT, которые используются при построении диаграмм функционального моделирования. Деятельность представляет собой некоторое действие или набор действий, которые имеют фиксированную цель и приводят к некоторому конечному результату. Иногда деятельность называют просто процессом. Модели IDEFO отслеживают различные виды деятельности системы, их описание и взаимодействие с другими процессами. На диаграммах деятельность или процесс изображается прямоугольником, который называется блоком. Стрелка служит для обозначения некоторого носителя или воздействия, которые обеспечивают перенос данных или объектов от одной деятельности к другой. Стрелки также необходимы для описания того, что именно производит деятельность и какие ресурсы она потребляет. Это так называемые роли стрелок — ICOM — сокращение первых букв от названий соответствующих стрелок IDEFO. При этом различают стрелки четырех видов:

- I (Input) - вход, то есть все, что поступает в процесс или потребляется процессом.
- C (Control) - управление или ограничения на выполнение операций процесса.
- O (Output) - выход или результат процесса.
- M (Mechanism) - механизм, который используется для выполнения процесса.

Методология IDEFO однозначно определяет, каким образом изображаются на диаграммах стрелки каждого вида ICOM. Стрелка Вход (Input) выходит из левой стороны рамки рабочего поля и входит слева в прямоугольник процесса. Стрелка Управление (Control) входит и выходит сверху. Стрелка Выход (Output) выходит из правой стороны процесса и входит в правую сторону рамки. Стрелка Механизм (Mechanism) входит в прямоугольник процесса снизу. Таким образом, базовое представление процесса на диаграммах IDEFO имеет следующий вид (рисунок 12.

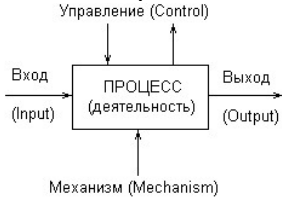


Рисунок 12 - Обозначение процесса и стрелок ICOM на диаграммах IDEF0

Техника построения диаграмм представляет собой главную особенность методологии IDEF-SADT. Место соединения стрелки с блоком определяет тип интерфейса. При этом все функции моделируемой системы и интерфейсы на диаграммах представляются в виде соответствующих блоков процессов и стрелок ICOM. Управляющая информация входит в блок сверху, в то время как информация, которая подвергается обработке, изображается с левой стороны блока. Результаты процесса представляются как выходы процесса и показываются с правой стороны блока. В качестве механизма может выступать человек или автоматизированная система, которые реализуют данную операцию. Соответствующий механизм на диаграмме представляется стрелкой, которая входит в блок процесса снизу.

Одной из наиболее важных особенностей методологии IDEF-SADT является постепенное введение все более детальных представлений модели системы по мере разработки отдельных диаграмм. Построение модели IDEF-SADT начинается с представления всей системы в виде простейшей диаграммы, состоящей из одного блока процесса и стрелок ICOM, служащих для изображения основных видов взаимодействия с объектами вне системы. Поскольку исходный процесс представляет всю систему как единое целое, данное представление является наиболее общим и подлежит дальнейшей декомпозиции.

Для иллюстрации основных идей методологии IDEF-SADT рассмотрим следующий простой пример. В качестве процесса будем представлять деятельность по оформлению кредита в банке. Входом данного процесса является заявка от клиента на получение кредита, а выходом — соответствующий результат, т. е. непосредственно кредит. При этом управляющими факторами являются правила оформления кредита, которые регламентируют условия получения соответствующих финансовых средств в кредит. Механизмом данного процесса является служащий банка, который

уполномочен выполнить все операции по оформлению кредита. Пример исходного представления процесса оформления кредита в банке изображен на рисунок 13.

В конечном итоге модель IDEF-SADT представляет собой серию иерархически взаимосвязанных диаграмм с сопроводительной документацией, которая разбивает исходное представление сложной системы на отдельные составные части. Детали каждого из основных процессов представляются в виде более детальных процессов на других диаграммах. В этом случае каждая диаграмма нижнего уровня является декомпозицией некоторого процесса из более общей диаграммы. Поэтому на каждом шаге декомпозиции более общая диаграмма конкретизируется на ряд более детальных диаграмм.

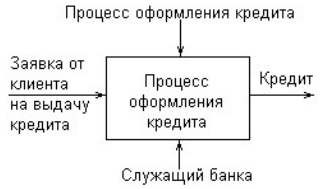


Рисунок 13 - Пример исходной диаграммы IDEF-SADT для процесса оформления кредита в банке

В настоящее время диаграммы структурного системного анализа IDEF-SADT продолжают использоваться целым рядом организаций для построения и детального анализа функциональной модели существующих на предприятии бизнес-процессов, а также для разработки новых бизнес-процессов. Основной недостаток данной методологии связан с отсутствием явных средств для объектно-ориентированного представления моделей сложных систем. Хотя некоторые аналитики отмечают важность знания и применения нотации IDEF-SADT, ограниченные возможности этой методологии применительно к реализации соответствующих графических моделей в объектно-ориентированном программном коде существенно сужают диапазон решаемых с ее помощью задач.

ДИАГРАММЫ ПОТОКОВ ДАННЫХ

Основой данной методологии графического моделирования информационных систем является специальная технология построения диаграмм потоков данных DFD. В разработке методологии DFD приняли участие многие аналитики, среди которых следует отметить Э. Йордона (E. Yourdon). Он является автором одной из первых графических нотаций DFD. В настоящее время наиболее распространенной является так называемая нотация Гейна-Сарсона (Gene-Sarson), основные элементы которой будут рассмотрены в этом разделе.

Модель системы в контексте DFD представляется в виде некоторой информационной модели, основными компонентами которой являются различные потоки данных, которые переносят информацию от одной подсистемы к другой. Каждая из подсистем выполняет определенные преобразования входного потока данных и передает результаты обработки информации в виде потоков данных для других подсистем.

Основными компонентами диаграмм потоков данных являются:

- внешние сущности
- накопители данных или хранилища
- процессы
- потоки данных
- системы/подсистемы

Внешняя сущность представляет собой материальный объект или физическое лицо, которые могут выступать в качестве источника или приемника информации. Определение некоторого объекта или системы в качестве внешней сущности не является строго фиксированным. Хотя внешняя сущность находится за пределами границ рассматриваемой системы, в процессе дальнейшего анализа некоторые внешние сущности могут быть перенесены внутрь диаграммы модели системы. С другой стороны, отдельные процессы могут быть вынесены за пределы диаграммы и представлены как внешние сущности. Примерами внешних сущностей могут служить: клиенты организации, заказчики, персонал, поставщики.

Внешняя сущность обозначается прямоугольником с тенью (рисунок 14), внутри которого указывается ее имя. При этом в качестве имени рекомендуется использовать существительное в именительном падеже. Иногда внешнюю сущность называют также терминатором.

Рисунок 14 - Изображение внешней сущности на диаграмме потоков данных

Процесс представляет собой совокупность операций по преобразованию входных потоков данных в выходные в соответствии с определенным алгоритмом или правилом. Хотя физически процесс может быть реализован различными способами, наиболее часто подразумевается программная реализация процесса. Процесс на диаграмме потоков данных изображается прямоугольником с закругленными вершинами (рисунок 15), разделенным на три секции

или поля горизонтальными линиями. Поле номера процесса служит для идентификации последнего. В среднем поле указывается имя процесса. В качестве имени рекомендовано использовать глагол в неопределенной форме с необходимыми дополнениями. Нижнее поле содержит указание на способ физической реализации процесса.

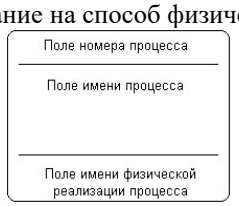


Рисунок 15 - Изображение процесса на диаграмме потоков данных

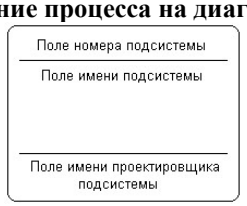


Рисунок 16 - Изображение подсистемы на диаграмме потоков данных

Информационная модель системы строится как некоторая иерархическая схема в виде так называемой контекстной диаграммы, на которой исходная модель последовательно представляется в виде модели подсистем соответствующих процессов преобразования данных. При этом подсистема или система на контекстной диаграмме DFD изображается так же, как и процесс — прямоугольником с закругленными вершинами (рисунок 16).

Накопитель данных или хранилище представляет собой абстрактное устройство или способ хранения информации, перемещаемой между процессами. Предполагается, что данные можно в любой момент поместить в накопитель и через некоторое время извлечь, причем физические способы помещения и извлечения данных могут быть произвольными. Накопитель данных может быть физически реализован различными способами, но наиболее часто предполагается его реализация в электронном виде на магнитных носителях. Накопитель данных на диаграмме потоков данных изображается прямоугольником с двумя полями (рисунок 17). Первое поле служит для указания номера или идентификатора накопителя, который начинается с буквы «D». Второе поле служит для указания имени. При этом в качестве имени накопителя рекомендуется использовать существительное, которое характеризует способ хранения соответствующей информации.

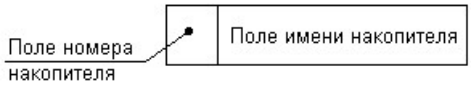


Рисунок 17 - Изображение накопителя на диаграмме потоков данных

Наконец, поток данных определяет качественный характер информации, передаваемой через некоторое соединение от источника к приемнику. Реальный поток данных может передаваться по сети между двумя компьютерами или любым другим способом, допускающим извлечение данных и их восстановление в требуемом формате. Поток данных на диаграмме DFD изображается линией со стрелкой на одном из ее концов, при этом стрелка показывает направление потока данных. Каждый поток данных имеет свое собственное имя, отражающее его содержание.

Таким образом, информационная модель системы в нотации DFD строится в виде диаграмм потоков данных, которые графически представляются с использованием соответствующей системы обозначений. В качестве примера рассмотрим упрощенную модель процесса получения некоторой суммы наличными по кредитной карточке клиентом банка. Внешними сущностями данного примера являются клиент банка и, возможно, служащий банка, который контролирует процесс обслуживания клиентов. Накопителем данных может быть база данных о состоянии счетов отдельных клиентов банка. Отдельные потоки данных отражают характер передаваемой информации, необходимой для обслуживания клиента банка. Соответствующая модель для данного примера может быть представлена в виде диаграммы потоков данных (рисунок 18).

В настоящее время диаграммы потоков данных используются в некоторых CASE-средствах для построения информационных моделей систем обработки данных. Основной недостаток этой методологии также связан с отсутствием явных средств для объектно-ориентированного представления моделей сложных систем, а также для представления сложных алгоритмов обработки данных. Поскольку на диаграммах DFD не указываются характеристики времени выполнения отдельных процессов и передачи данных между процессами, то модели систем, реализующих синхронную обработку данных, не могут быть адекватно представлены в нотации DFD. Все эти особенности методологии структурного системного анализа ограничили возможности ее широкого применения и послужили основой для включения соответствующих средств в унифицированный язык моделирования.

Рисунок 18 - Пример диаграммы DFD для процесса получения некоторой суммы наличными по кредитной карточке