Министерство образования Белгородской области
Областное государственное автономное профессиональное образовательное учреждение
«Белгородский индустриальный колледж»

Рассмотрено
предметно-цикловой комиссией
Протокол заседания № ____
От «____»_____ 2022 г.
Председатель цикловой комиссии
_____/ Третьяк И.Ю.

МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ

МДК 05.03. Тестирование информационных систем

для специальности

09.02.07 «Информационные системы и программирование»

Разработчик:
Шершнева М.А. преподаватель специальных дисциплин ОГАПОУ БИК

Белгород 2022

Тематика лабораторных работ по МДК 05.03. Тестирование информационных систем

Номер лабораторной работы	Тема лабораторной работ	Кол-во часов
1.	Лабораторная работа «Разработка тестового сценария проекта»	2
2.	Лабораторная работа «Ручной подход. Ручное тестирование и подход генерации тестовых наборов при разработке тестов»	2
3.	Лабораторная работа «Разработка тестовых пакетов»	2
4.	Лабораторная работа «Использование инструментария анализа качества»	2
5.	Лабораторная работа «Анализ и обеспечение обработки исключительных ситуаций»	2
6.	Лабораторная работа «Функциональное тестирование»	2
7.	Лабораторная работа «Тестирование безопасности»	2
8.	Лабораторная работа «Тестирование безопасности»	2
9.	Лабораторная работа «Нагрузочное тестирование, стрессовое тестирование»	2
10.	Лабораторная работа «Модульное тестирование»	2
11.	Лабораторная работа «Модульное тестирование»	2
12.	Лабораторная работа «Тестирование интеграции»	2
13.	Лабораторная работа «Тестирование интеграции»	2
14.	Лабораторная работа «Системное тестирование»	2
15.	Лабораторная работа «Системное тестирование»	2
16.	Лабораторная работа «Конфигурационное тестирование»	2
17.	Лабораторная работа «Конфигурационное тестирование»	2
18.	Лабораторная работа «Тестирование установки»	2
19.	Лабораторная работа «Методы автоматизации исполнения тестов»	2
20.	Лабораторная работа «Автоматизация тестирования с помощью скриптов»	2
21.	Лабораторная работа «Автоматизация тестирования с помощью скриптов»	2
22.	Лабораторная работа «Автоматическая генерация тестов на основе формального описания»	2
23.	Лабораторная работа «Автономная отладка ИС»	2
24.	Лабораторная работа «Комплексная отладка ИС»	2
25.	Лабораторная работа «Поиск ошибок в программах. Классификация ошибок и тестов»	2
26.	Лабораторная работа «Моделирование бизнес-процессов в ИС»	2
27.	Лабораторная работа «Моделирование бизнес-процессов в ИС»	2
ИТОГО:		

Лабораторная работа № 1
Разработка тестового сценария проекта

Цель работы: проведение подготовительных работ для тестирования программного обеспечения. Подготовительные работы включают планирование тестирования программного обеспечения - создание плана тестирования программного обеспечения (ПО).

Задание: Разработать план тестирования программы.

Пререквизиты (исходные данные)

Студент составляет ТЗ на программу. Разрабатывает ПО, для которого будет создаваться план. Для разработки может быть выбран любой язык программирования.

Предварительные (теоретические) сведения

План тестирования – это документ, который систематизированно писывает, как выбранное ПО будет тестироваться. Существует большое количество шаблонов плана тестирования, в общем они все похожи. Один из них – это шаблон IEEE 829-2008. Доступные шаблоны плана тестирования имеют незначительные различия, но в основном состоят из следующих разделов:

1. Область тестирования - определяется область тестирующей деятельности.
2. Стратегии тестирования - какие стратегии тестирования должны использоваться во время тестирования программы.
3. Начальные условия (пререквизиты) - какие задачи должны решены перед началом тестирования.
4. Приоритеты тестирования – какие тестирущие действия, задачи и программные компоненты имеют более высокий приоритет для тестирования.
5. Методики тестирования - какие методики тестирования будут использоватья в ходе тестирования.
6. Роли и обязанности - определяются члены команды и их обязанности для определенных частей процесса тестирования.
7. Результаты - что является ожидаемыми результатами тестирования, какие задачи должны быть решены, какие документы и отчеты, должны быть подготовлены.
8. Среда тестирования - какая необходима среда тестирования. Определяется необходимая конфигурация программного и аппаратного обеспечения, чтобы начать тестирование.
9. Сценарии тестов - определяются сценарии тестов, какие действия тестировщик должен выполнить, какие тесты должны юыть запрограммированы.
10. Управление испытаниями (тестами) - определяется как управлять созданными тестами, сценариями, управляются, как увидеть выявленные дефекты и записать их. Как выполняется регрессионное тестирование.
11. Графики испытаний - определяется график тестирующих действий.

12. Риски тестирования - что является возможными рисками, которые могли бы помешать выполнению плана тестирования, определяются планы, как избежать их или смягчить последствия риска.

Задания для лабораторной работы:

1. Подготовьте на 1 страницу описание ее архитектуры, функций, особенности, компоненты тестируемой программы.

2. Студент может выбрать шаблон плана тестирования, какой он предпочитает. Один из шаблонов плана представлен в конце описания лабораторной работы.

3. План испытания настраивается путем добавления нужных разделов и удаления ненужных.

4. Документ плана испытаний создается заполнением пустых разделов шаблона плана.

Защита работы:

1. Должен быть представлен плана тестирования в виде документа.

2. Студент должен быть готов ответить на вопросы, касающиеся выполненной работы.

Вопросы студентам для изучения (подготовка к защите):

1. Какова цель создания плана тестирования?

2. Когда предполагается, что тест выполнен?

3. В какое время в ходе разработки программного обеспечения лучше всего выполнять тестирующие действия?

Используемые источники:

1. IEEE 829-2008. IEEE Standard for Software and System Test Documentation.

Пример плана тестирования:

Введение

Этот документ описывает план тестирования для системы обработки заказов ABC. Полная стратегия тестирования программного обеспечения состоит из следующих типов испытаний и выполнеяется в следующем порядке:

1. Тестирование компонентов (модульное тестирование). Тестируются все программные компоненты (при этом проверяется покрытие кода тестами). Параллельно проводится анализ кода.

2. Тестирование интерфейсов пользователя.

2. Тестирование интеграции. Тестируется программное обеспечение, чтобы гарантировать, что компоненты взаимодействуют правильно.

3. Проверка правильности (валидация). Проводится тестирование программного обеспечения в эмулированной среде производства, чтобы проверить ее функциональные возможности.

4. Приемочные испытания. Проводятся приемочные испытания, чтобы гарантировать, что функциональные возможности ПО соответствуют ожиданиям конечного пользователя.

Приемочные испытания состоят из заключительного набора тестов, которые выполняются перед официальным запуском программной системы.

Область тестирования

Премочные испытания состоят из:

1. Веб-системы ABC, версия V1.

2. Моделей вариантов использования для ролей клиента, менеджера, администратора.

Цель этого тестирования состоит в том, чтобы определить, насколько хорошо внедряемое программное обеспечение соответствует функциональным требованиям, также идентифицировать обнаруженные проблемы и обеспечить их фиксацию. Тестирование также позволило бы собрать тестовые данные и тестовые результаты для дальнейшего регрессионного тестирования ПО в течение периода его поддержки.

Стратегии тестирования

Главное в приемочном испытании это продемонстрировать, что программное обеспечение и его инфраструктура устойчивы, и работают надежно. Все другие тестирующие действия должны быть закончены перед стартовой фазой приемочного испытания. Приемочное испытание ориентировано на тестирование программного обеспечения с точки зрения пользователя, чтобы проверить, как программное обеспечение используется день ото дня и как оно соответствует определенным требованиям качества. В течение приемочного испытания должен присутствовать представитель от компании клиента.

Дополнительные действия тестирования должны включать: модульное тестирование, тестирование интеграции, безопасности и производительности.

Начальные условия (пререквизиты)

Задачи, которые должны быть решены перед началом тестирующей деятельности:

1. Имеется законченная программная спецификация в виде моделей вариантов использования и сценариев использования;
2. Работающее реализованное программное обеспечение,
3. Установленная процедура фиксации обнаруженных проблем в течение испытания,
4. Набор определенных вариантов использования для приемочного испытания, чтобы проверить все функциональные возможности программы,
5. Выбранная среда тестирования,
6. *Распределенные (сервера)* тестовые ресурсы,

7. Определенные стандарты приемочных испытаний.

Приоритеты тестирования

Следующие проверки перечислены в порядке уменьшения приоритетного уровня (первое имеет самый высокий приоритет):

1. Функции - все ли заданные функции программы выполняются как ожидалось?
2. Удобство и простота использования - действительно ли программное обеспечение является дружественным по отношению к пользователю?
3. Защита - данные защищены?
4. Выполнение – программное обеспечение соответствует согласованному критерию выполнения? (что в первую очередь проверять?????)

Методы тестирования

Будут использоваться следующие методы тестирования:

1. Тестовые сценарии – сценарии вариантов использования (с предопределенным вводом и ожидаемыми выходными данными).
2. Тестовые сценарии без данных - тестировщики выберут входные данные во время испытания.
3. Тесты на несанкционированное использование – сценарии действий –попыток получить несанкционированный доступ к данным ПО.
4. Проверка на удобство использования ПО - действия, чтобы оценить простоту системы в использовании.
5. Статистика выполнения (статистическое тестирование)- набор характеристик работы и сравнение с заданными параметрами.
6. Стрессовое тестирование.
7. Нагрузочное тестирование (профилирование и производительность)
8. Тестирование БД (SQL-инъекции и тестирование хранимых процедур)
9. Кодированное тестирование интерфейса.

Управление тестами

Определяются следующие роли и обязанности:

1. Лидер проверки качества - человек, ответственный за процесс планирования тестирования и его выполнение.
2. Тестировщик - выполняет действия по тестированию, определенные в плане тестирования.
3. Менеджер по ПО - гарантирует, что тесты выполняются успешно с точки зрения пользователя.
4. Поддержка тестирования - гарантирует, что техническое оборудование на месте и помогает в течение испытаний.

Результаты

После тестирования должны быть в наличие следующие документы:

1. План тестирования- этот документ со всеми изменениями, сделанными в ходе процесса тестирования.

2. Запросы на изменение - документ, описывающий изменения ПО, вызванные изменением требований или обнаруженными дефектами в течение испытания.

3. Еженедельные отчеты о ходе тестирования.

4. Заключительный отчет, подписанный заказчиком, подтверждающий, что система отвечает всем функциональным требованиям и требованиям качества.

Лабораторная работа № 2
Ручное тестирование, генерация тестов

Цель работы: Овладение навыками ручного тестирования и составление
тестовых случаев.

Общие сведения

Ручное тестирование заключается в выполнении задокументированной процедуры, где описана методика выполнения тестов, задающая порядок тестов и для каждого теста - список значений параметров, который подается на вход, и список результатов, ожидаемых на выходе. Поскольку процедура предназначена для выполнения человеком, в ее описании для краткости могут использоваться некоторые значения по умолчанию, ориентированные на здравый смысл, или ссылки на информацию, хранящуюся в другом документе.

Описание тестов разрабатывается для облегчения анализа и поддержки тестового набора. Описание может быть реализовано в произвольной форме, но при этом должны выполнять следующие задачи:

1.Анализировать степень покрытия продукта тестами на основании описания тестового набора.

2.Для любой функции тестируемого продукта найти тесты, в которых функция используется.

3.Для любого теста определить все функции и их сочетания, которые данный тест использует (затрагивает).

4.Понять структуру и взаимосвязи тестовых файлов.

5.Понять принцип построения системы автоматизации
тестирования

Задание

Подготовить тестовый случай, выполнить и задокументировать результаты.

Лабораторная работа № 3
Тема: Разработка тестовых пакетов

Цель: изучить методы тестирования и отладки на практическом примере.

Краткая теория и ход выполнения работы

1) Мето
ды тестирования и отладки

Тестирование—процесс выполнения программ с целью обнаружения факта наличия ошибок.

Отладка—процесс локализации и устранения ошибок.

Процессы тестирования и отладки схематически могут быть представлены следующим образом:

рис. 1 Тестирование

Тестирование начинается с разработки множества тестов и их исполнения на основе одной из выбранных методик. Подготовка дополнительных тестов потребуется при недостаточной полноте тестирования, невозможности локализовать проблему с помощью имеющихся тестов и необходимости выполнить контроль сделанного исправления.

Существуют две основные стратегии тестирования. Тестирование программы как черного ящика, при котором программа рассматривается как объект, внутренняя структура которого неизвестна.

Тестирование программы как прозрачного (белого) ящика подразумевает знание исходного кода программы и полный доступ к нему.

При тестировании по типу «черного ящика» тесты демонстрируют:

♣как выполняются функции программы;

♣как принимаются исходные данные;

♣как вырабатываются результаты;

♣как сохраняется целостность внешней информации.

При тестировании «черного ящика» рассматриваются системные характеристики программ, игнорируется их внутренняя логическая структура.

Для тестирования программ методом «черного ящика» готовятся определенные группы тестов.

♣Для тестирования классов эквивалентностей. Классы эквивалентности позволяют вместо большого количества тестов использовать лишь их небольшое подмножество. Каждый тест представляет набор тестов, на которых программа ведет себя одинаково. Существует два типа классов эквивалентностей:

¬Класс корректных тестовых случаев, отражающих типичную «нормальную» ситуацию.

¬Класс тестов, содержащих ненормальную ситуацию, т.е. описывающих ситуацию, которой быть не должно.

♣Для тестирования граничных значений.

♣Для анализа причинно–следственных связей.

♣Для тестирования тех утверждений, которые приводятся в документации.

При тестировании по типу «белого ящика» исследуются внутренние элементы и связи между ними.

Объектом тестирования является не внешнее, а внутреннее поведение программы. Проверяется корректность построения всех элементов программы и правильность их взаимодействия друг с другом. Тестирование по принципу «белого ящика» характеризуется степенью, в какой тесты выполняют или покрывают логику (исходный текст) программы.

Наиболее важный принцип, относящийся к тестированию программ, состоит в том, чтобы думать об этой стадии еще на этапе написания программы. Следует постоянно задаваться вопросом: как будет тестироваться данный сегмент? Если ответ на вопрос о способе тестирования программы неясен, она должна быть либо переписана заново, либо разбита на модули.

Для составления тестов используются следующие источники:

♣справочники;

♣вычисления вручную;

♣использование результатов, полученных при помощи другой программы.

Поскольку в процессе разработки приходится тестировать еще не завершенную программу, все подходы делятся на две группы.

Тестирование сверху вниз. Применяется, если программа разрабатывается сверху вниз. В данном случае используются «заглушки» — фрагменты кода, имитирующие еще не написанные части программы.

Тестирование снизу вверх. При этом, как правило, дополнительно должна быть создана программа —«драйвер», организующая взаимодействие уже написанных модулей.

Если процесс тестирования показал, что программа работает неправильно, то начинается процесс отладки.

В процессе отладки локализуется ошибка.

Отладка программы —процесс творческий и плохо формализуемый. Тем не менее, основной идее отладке можно придать вид следующего алгоритма:

♣Следует начать с изучения уже доступных исходных и результирующих данных.

♣Сформулировать некоторую гипотезу, которая объясняет получение таких результирующих данных.

♣Подготовить новые исходные данные и провести эксперимент, который позволит доказать или опровергнуть гипотезу.

Для отладки программ в инструментальные среды программирования встраиваются специальные отладчики.

2)Средства отладки Delphi

Отладка—это локализация ошибки и ее исправление. Для этого используются точки прерывания, пошаговое выполнение программы и просмотр ряда переменных на различных шагах выполнения программы.

Окно наблюдения

Наблюдать за состоянием переменной или выражения можно с помощью специального окна, вызываемого опцией

View | Debug windows | Watches

(cm. там же).

Окно наблюдения (рис.2) используется в отладочном режиме для наблюдения за изменением значений выражений, помещенных в это окно. Для добавления нового выражения щелкните по окну правой кнопкой мыши и выберите опцию

`New Watch.`

В строке `Expression` введите выражение. Окно `Repeat count` определяет количество показываемых элементов массивов данных; окно

`Digits` указывает количество значащих цифр для отображения вещественных данных; переключатель `Enabled` разрешает или запрещает вычисление выражения. Остальные элементы определяют вид представления значения.

В последних версиях Delphi вы можете просмотреть в отладочном режиме текущее значение любой переменной, если укажите на нее курсором: значение появится в ярлычке рядом с курсором.

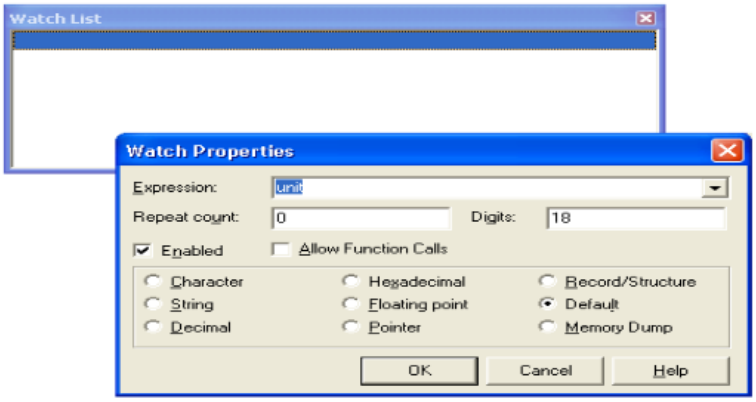


Рис. 2 Точки контрольного останова

Точка контрольного останова определяет оператор в программе, перед выполнением которого программа прервет свою работу и управление будет передано среде Delphi. Точка останова задается с помощью опции view | Debug windows | Breakponts.

Окно точек останова (рис. 2) содержит список всех установленных в проекте точек, перед выполнением которых происходит прекращение работы программы и управление получает среда Delphi.

Для добавления новой точки следует щелкнуть по окну правой кнопкой мыши и выбрать опцию Add.

В этом случае появляется окно, с помощью которого можно указать положение добавляемой точки:

FileName-определяет имя файла; Line number-номер строки от начала файла (в момент появления окна оно содержит файл и строку с текстовым курсором). В строке COndition можно указать условие останова в виде логического выражения (например, MyValue = Max-Value-12), а в строке Pass count-количество проходов программы через контрольную точку без прерывания вычислений.

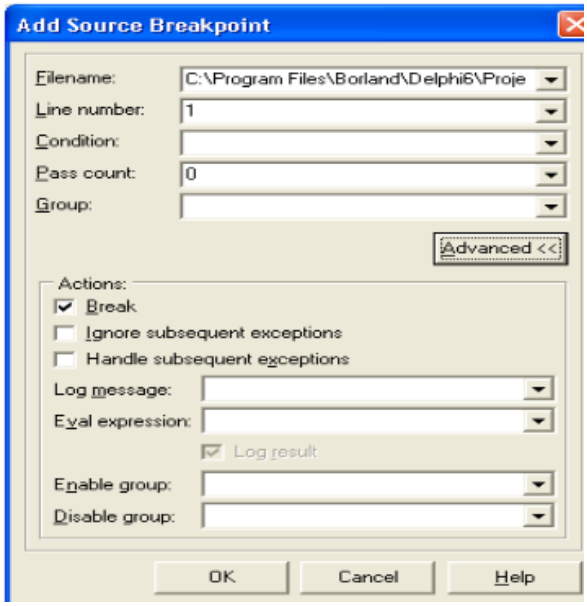


рис. 3 Окно точек останова (слева) и окно добавления новой точки (справа)

В Delphi 5 и 6 с любой точкой можно связать одно или несколько действий. Для этого нужно активизировать окно точек останова, вызвать его локальное меню (щелчок правой кнопкой) и выбрать продолжение Properties. В появившемся окне свойств щелкнуть по кнопке Advanced (рис. П1.19).

В нижней части окна имеется панель Actions, с помощью которой и определяются действия для точки останова, указанной в верхней части окна.

Break-простой останов перед выполнением помеченного оператора. ignore subsequent exceptions-если переключатель установлен, игнорируются все возможные последующие исключения в текущем отладочном сеансе до очередной точки останова, в которой, возможно, это действие будет отменено.

Handle subsequent exceptions-после установки этого переключателя отменяется действие предыдущего переключателя и возобновляется обработка возможных исключений.

С помощью Log message вы можете указать произвольное сообщение, связанное с точкой останова, а с помощью Eval expression-вычислить некоторое выражение и поместить его результат в это сообщение.

Трассировка программы

Перед исполнением оператора, в котором установлена точка контрольного останова, работа программы будет прервана, управление получит среда Delphi, а в окне наблюдения отразится текущее значение

наблюдаемы переменных и/или выражений. Теперь программист может прослеживать работу программы по шагам с помощью клавиш F7 и F8 или инструментальных кнопок. При нажатии F8 будут выполнены запрограммированные в текущей строке действия, и работа программы прервется перед выполнением следующей строки текста программы.

Контрольная точка останова выделяется по умолчанию красным цветом, а текущая прослеживаемая строка -синим. Если программа остановлена в контрольной точке, т.е. когда текущая строка совпадает со строкой останова, строка выделяется красным цветом, Признаком текущей строки является особое выделение строки в служебной зоне слева в окне редактора (рис. 4).

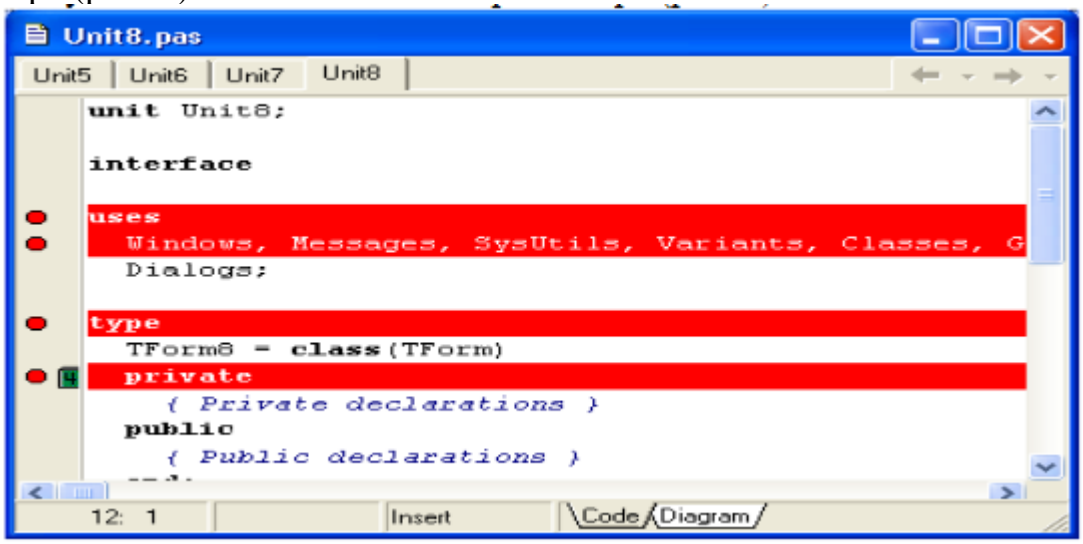


Рис. 4 Фрагмент окна редактора в режиме отладки

Кстати, чтобы установить/снять точку контрольного останова, достаточно щелкнуть мышью по служебной зоне слева от нужной строки или установить в эту строку текстовый курсор и нажать F5.

При нажатии F7 среда выполняет те же действия, что и при нажатии F8, однако, если в текущей строке содержится вызов подпрограммы пользователя, программа прервет свою работу перед выполнением первого исполняемого оператора этой подпрограммы, т.е. клавиша F7 позволяет прослеживать работу вызываемых подпрограмм.

После трассировки нужного фрагмента программы можно продолжить нормальную ее работу, нажав клавишу F9.

3) Самостоятельная работа

1.Разработайте в среде Delphi7 проект в соответствии с вариантом.

2.Разработайте план тестирования Вашего программного комплекса. При подготовке контрольных тестов воспользуйтесь вышеописанными рекомендациями. Тесты должны отображать:

a.типичную ситуацию;

b.ненормальную ситуацию;

c.граничные значения;

d.затрагивающие причинно–следственные связи.

Одним из тестов является контрольный пример из раздела «Постановка задачи.

3.Воспользуйтесь опцией GotoCursorдля перехода в режим отладки. При остановке выполнения программы добавьте в окно просмотра (AddWatch) наименования нескольких интересующих Вас переменных. Вызовите окно просмотра (Watch).

Далее выполняйте программу по шагам (F7 или F8). В окне просмотра можно наблюдать интересующие Вас переменные. Если окно просмотра по какой-то причине исчезло с экрана, в него можно перейти с помощью клавиши F6. Для прекращения работы программы следует нажать комбинацию клавиш CtrlF2 или выполнить опцию ProgramReset. Можно продолжить выполнение программы, нажав CtrlF9.

4.Установите в программе несколько контрольных точек и запустите программу на выполнение. Просмотрите значения интересующих Вас переменных с помощью окна просмотра, так же как это было предложено в предыдущем пункте.

5.Воспользуйтесь опцией GotoCursor для перехода в режим отладки. При остановке выполнения программы откройте окно Evaluate/modify

и просмотрите значения интересующих Вас переменных. Измените значение какой-нибудь переменной, записав новое в окно NewValue.

6.Выполните такое же задания, используя точки останова (Breakpoint).

7.Ознакомьтесь с процессом выполнения программы при запуске ее с помощью опций TraceInto(F7) и StepOver(F8).

8.Продолжите отладку разрабатываемой программы, используя навыки работы с отладчиком.

Отчет должен содержать:

1. № и тему лабораторной работы.
2.Задание по лабораторной работе.
3.Листинг программы.
4.Скриншоты выполнения программы.
5.Ответы на контрольные вопросы.

Контрольные вопросы:

1.Что такое тестирование?
2.Что такое отладка?
3.Методы тестирования

Лабораторная работа №5
Анализ и обеспечение обработки исключительных ситуаций

1 Цель и порядок работы

Цель работы - изучить операторы, используемые при обработке исключительных ситуаций, возникающих во время выполнения вычислительных процессов, получить практические навыки в составлении программ.

Порядок выполнения работы:

- ознакомиться с описанием лабораторной работы;
- получить задание у преподавателя по вариантам;
- написать программу, ввести программу, отладить и решить ее на ЭВМ;
- оформить отчет.

2 Общие сведения

- Исключительная ситуация (или исключение) - это ошибка, которая возникает во время выполнения программы. Используя C#-подсистему обработки исключительных ситуаций, с такими ошибками можно справляться. В C# эта подсистема включает в себя усовершенствованные методы, используемые в языках C++ и Java. Преимущество подсистемы обработки исключений состоит в автоматизации создания большей части кода, который ранее необходимо было вводить в программы "вручную". Обработка исключений упрощает "работу над ошибками", позволяя в программах определять блок кода, именуемый обработчиком исключении, который будет автоматически выполняться при возникновении определенной ошибки. В этом случае не обязательно проверять результат выполнения каждой конкретной операции или метода вручную. Если ошибка возникнет, ее должным образом обработает обработчик исключений.

Еще одним преимуществом обработки исключительных ситуаций в C# является определение стандартных исключений для таких распространенных программных ошибок, как деление на нуль или попадание вне диапазона определения индекса. Чтобы отреагировать на возникновение таких ошибок, программа должна отслеживать и обрабатывать эти исключения. Без знания возможностей C#-подсистемы обработки исключений успешное программирование на C# попросту невозможно.

В C# исключения представляются классами. Все классы исключений должны быть выведены из встроенного класса исключений Exception, который является частью пространства имен System. Таким образом, все исключения - подклассы класса Exception.

Программные инструкции, которые нужно проконтролировать на предмет исключений, помещаются в *try*-блок. Если исключение возникает в этом блоке, оно дает знать о себе выбросом определенного рода информации. Это выброшенное исключение может быть перехвачено программным путем с помощью *catch*-блока и обработано соответствующим образом. Системные

исключения автоматически генерируются C#-системой динамического управления. Чтобы сгенерировать исключение вручную, используется ключевое слово *throw*. Любой код, который должен быть обязательно выполнен при выходе из *try*-блока, помещается в блок *finally*.

Ядром обработки исключений являются блоки *try* и *catch*. Эти ключевые слова работают "в одной связке"; формат записи *try/catch*-блоков обработки исключений имеет следующий вид:

```
try {
// Блок кода, подлежащий проверке на наличие ошибок.
}
catch (ExcepTypel exOb) {
// Обработчик для исключения типа ExcepTypel
}
catch (ExcepType2 exOb) {
// Обработчик для исключения типа ExcepType2
}
```

Здесь ExcepType - это тип сгенерированного исключения. После "выброса" исключение перехватывается соответствующей инструкцией *catch*, которая его обрабатывает. Как видно из формата записи *try/catch*-блоков, с *try*-блоком может быть связана не одна, а несколько catch-инструкций. Какая именно из них будет выполнена, определит тип исключения. Другими словами, будет выполнена та *catch*-инструкция, тип исключения которой совпадает с типом сгенерированного исключения (а все остальные будут проигнорированы). После перехвата исключения параметр exOb примет его значение.

Задавать параметр exOb необязательно. Если обработчику исключения не нужен доступ к объекту исключения (как это часто бывает), в задании параметра exOb нет необходимости. Поэтому во многих примерах этой главы параметр exOb не задан.

Важно понимать следующее: если исключение не генерируется, то *try*-блок завершается нормально, и все его *catch*-инструкции игнорируются. Выполнение программы продолжается с первой инструкции, которая стоит после последней инструкции *catch*. Таким образом, *catch*-инструкция (из предложенных после *try*-блока) выполняется только в случае, если сгенерировано соответствующее исключение.

Ниже представлены наиболее часто используемые исключения, определенные в пространстве имен System:

Исключение	Значение
ArrayTypeMismatchException	Тип сохраняемого значения несовместим с типом массива
DivideByZeroException	Попытка деления на нуль
IndexOutOfRangeException	Индекс массива оказался вне диапазона
InvalidCastException	Неверно выполнено динамическое приведение типов

OutofMemoryException	Недостаточный объем свободной памяти
OverflowException	Имеет место арифметическое переполнение
NullReferenceException	Попытка использовать нулевую ссылку
StackoverflowException	Переполнение стека

Иногда требуется перехватывать все исключения, независимо от их типа. Для этого используется *catch*-инструкция без параметров. В этом случае создается обработчик "глобального перехвата", который используется, чтобы программа гарантированно обработала все исключения.

Задание: оптимизировать программу, производящую простые арифметические операции над числами (сложение, вычитание, умножение и деление), используя обработку исключительных ситуаций (обработка ввода чисел и операции деления).

Пример:

```
using System;
namespace ConsoleApplication {
 class OurClass {
   static void Main(string[] args) {
 float num1 = 1, num2 = 2, summarize, multiply, sub, divide = 0;
 Console.Write("Введите первое число:");
 try { num1 = float.Parse(Console.ReadLine()); }
 catch {
  Console.WriteLine("Неправильный формат числа!\n"+
      "В качестве значения первого числа будет 1");
}
Console.Write("Введите второе число:");
 try { num2 = float.Parse(Console.ReadLine()); }
    catch {
  Console.WriteLine("Неправильный формат числа!\n"+
      "В качестве значения второго числа будет 2");
 }
summarize = num1 + num2; multiply = num1 * num2; sub = num1 - num2;
 try { divide = num1 / num2; }
 catch(DivideByZeroException) {
  Console.WriteLine("Нельзя делить на нуль!");
 }
    Console.WriteLine(
    "\n" + num1 + " + " + num2 + " = " + summarize +
"\n" + num1 + " * " + num2 + " = " + multiply +
"\n" + num1 + " - " + num2 + " = " + sub +
"\n" + num1 + " / " + num2 + " = " + divide);
Console.Write("\nДля выхода из программы нажмите [Enter]:");
    string anykey = Console.ReadLine();
  }
 }
```

}

3. Задание для самостоятельной работы

Для своего варианта задания лабораторной работы № 2 оптимизировать программу, включив в нее обработку исключительных ситуаций: ввода данных, операции деления и др.

1. Содержание отчета

4.1. Титульный лист.

4.2. Краткое теоретическое описание.

4.3. Задание на лабораторную работу, включающее математическую формулировку задачи.

Лабораторная работа № 6
Функциональное тестирование

Цель работы: 1. Изучение назначения и задач функционального тестирования.

2. Программная реализация тестов, производящих функциональное тестирование алгоритма пирамидальной сортировки из курса лабораторных работ.

Общие сведения

Функциональное тестирование — это тестирование ПО в целях проверки реализуемости функциональных требований, то есть способности ПО в определённых условиях решать задачи, нужные пользователям. Функциональные требования определяют, что именно делает ПО, какие задачи оно решает.

Для предоставленной реализации алгоритма пирамидальной сортировки из курса лабораторных работ необходимо установить:

- выполняет ли она возложенные на неё функции для всех возможных исходных данных (целых положительных, отрицательных чисел и нуля), то есть является ли массив, обработанный программой, отсортированным;
- соответствует ли она заявленной теоретической трудоёмкости (см. разделы 1.2, 1.8), то есть не будет ли программа исполняться дольше обещанного при возрастании объёма входных данных.

Для того чтобы определить тип зависимости в экспериментальных данных, необходимо понять, какой функцией от объёма исходных данных ограничена экспериментальная зависимость, то есть найти её асимптотическую оценку («O» большое).

Запись вида $f(n) = O(g(n))$ означает, что функция $f(n)$ возрастает медленнее, чем функция $c*g(n)$ при $c = c_1$ и $n = N$, где c_1 и N могут быть сколь угодно большими числами, то есть при

$c = c_1$ и $n \geq N$, $c*g(n) \geq f(n)$ (4.1).

Функция $g(n)$ является минимальной по степени, удовлетворяющей условию (4.1).

Юнит-тестирование

Модульное тестирование, или юнит-тестирование (англ. unit testing) — процесс в программировании, позволяющий проверить на корректность отдельные модули исходного кода программы.

Идея состоит в том, чтобы писать тесты для каждой нетривиальной функции или метода. Это позволяет достаточно быстро проверить, не привело ли очередное изменение кода к регрессии, то есть к появлению ошибок в уже оттестированных местах программы, а также облегчает обнаружение и устранение таких ошибок. В данной работе мы будем использовать юнит-тесты для проверки функциональных требований программы.

Для использования юнит-тестов будем использовать JUnit. JUnit — библиотека для модульного тестирования программного обеспечения на

языке Java. Скачать свежую версию данной библиотеки можно по адресу http://www.junit.org.

Пример юнит-теста, проверяющего равенство 2+2=4, приведен в листинге 4.1.

Листинг 4.1 – пример юнит-теста

```
import org.junit.Test;
import junit.framework.Assert;
public class MathTest {
@Test
public void testEquals() {
Assert.assertEquals(4, 2 + 2);
Assert.assertTrue(4 == 2 + 2);
}
@Test
public void testNotEquals() {
Assert.assertFalse(5 == 2 + 2);
}
}
```

Предварительная подготовка к работе

Так как алгоритм реализован на языке Java, то юнит-тесты следует писать с использованием библиотеки JUnit (http://www.junit.org). В качестве седы разработки можно использовать любую удобную Вам IDE (Eclipse / NetBeans / др.). Реализовать юнит-тесты следует так, чтобы они выполнял функциональное тестирование программы сортировки в соответствии с вышеуказанными задачами.

Юнит-тест на проверку теоретической трудоёмкости представляет собой получение времени работы программы сортировки на исходных данных объёмом от 1 до 10000 элементов. Чтобы получить достоверные данные следует провести эксперимент для каждого из значений объема исходных данных не менее 5 раз. Замер времени удобно проводить с помощью функции System.*nanoTime*().

Получив результаты теста, необходимо произвести их статистическую обработку. Следует вычислить медиану по всем экспериментам для каждого значения объема исходных данных. Это делается для того, чтобы получить данные о времени работы программы без случайных выбросов. Такие выбросы могут возникнуть вследствие занятости операционной системы, так как она, скорее всего, не является системой реального времени и в некоторых ситуациях не способна предоставить программе требуемые ресурсы с минимальными задержками.

Получив отфильтрованные данные, необходимо найти наиболее подходящую им регрессионную модель (линию тренда). Это можно сделать, например, в программе для работы с электронными таблицами MS Excel. Далее следует подобрать асимптотическую оценку этой модели, которая будет являться экспериментальной трудоёмкостью проверяемой программы сортировки.

Обработка результатов эксперимента

Для удобства выполнения статистической обработки данных все полученные значения необходимо перенести в MS Excel так, чтобы столбец содержал результаты одного эксперимента для объемов исходных данных от 1 до 10000.

Для того, чтобы избавиться от случайных выбросов вычислим медиану для каждого эксперимента, воспользовавшись встроенной функцией MS Excel (4.2). Медиана — это возможное значение, которое делит отсортированный ряд чисел на две равные части: 50 % «нижних» единиц ряда данных будут иметь значение не больше, чем медиана, а «верхние» 50 % — значения не меньше, чем медиана. В нашем случае, можно сказать, что медиана — это то значение времени, за которое выполнился алгоритм в среднестатистическом эксперименте.

МЕДИАНА(число 1;[число2];…) (4.2)

Данная функция вычисляет медиану для выбранного в качестве аргумента диапазона ячеек или чисел.

Результатом этого шага обработки данных станет столбец из 10000 элементов с медианно отфильтрованными экспериментальными данными (Рисунок 4.1).

	B	C	D	E	F	G	H	I	J
1	3632	3073	2793	2794	2794		2933,5		
2	3632	3631	3631	3631	3632		=МЕДИАНА(A2:F2)		
3	4749	4470	4749	4749	4749		МЕДИАНА(**число1**; [число2]; …)		
4	6146	6146	6146	6146	5867		6146		

Рисунок 4.1 – Заполнение столбца значениями медианы экспериментальных данных

После получения среднестатистических данных о трудоемкости выполнения алгоритма сортировки на разных объемах данных, необходимо построить их график (рисунок 4.2) и добавить на него линию тренда (рисунок 4.3).

Рисунок 4.2 – График экспериментальных данных

Для того чтобы добавить линию тренда, необходимо выделить данные на графике, нажать на правую кнопку мыши и выбрать пункт «Добавить линию тренда».

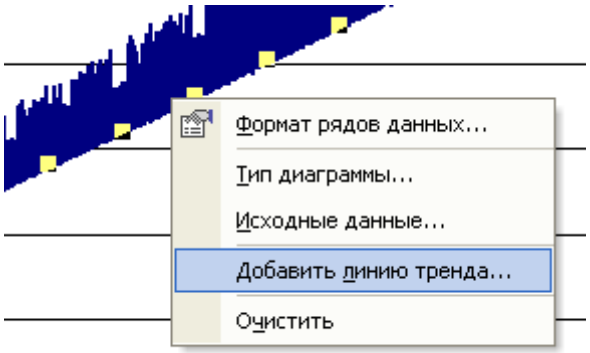


Рисунок 4.3 – Добавление линии тренда

В появившемся меню необходимо зайти на вкладку «Параметры» и выделить пункты:

1. Пересечение кривой с осью Y в точке: (поставить значение из первой ячейки столбика медиан, если это возможно)
2. Показывать уравнение на диаграмме
3. Поместить на диаграмму величину достоверности аппроксимации (R^2)

Необходимо опробовать все предлагаемые на вкладке «Тип» модели и выбрать ту, показатель достоверности аппроксимации (R^2) которой будет наибольшим. Следует выписать уравнение этой модели с диаграммы.

Далее нужно найти асимптотическую оценку регрессионной модели («О большое»). Для этого необходимо проверить, как ведет себя уравнение модели относительно основных классов сложности на больших n.

Основные классы сложности приведены в таблице 4.1:

Таблица 4.1 – Основные классы сложности

$f(n) = O(1)$	константа
$f(n) = O(\log(n))$	логарифмический рост
$f(n) = O(n)$	линейный рост
$f(n) = O(n*\log(n))$	квазилинейный рост
$f(n) = O(n^m)$	полиномиальный рост
$f(n) = O(2^n)$	экспоненциальный рост

Асимптотической оценкой $f(n) = O(g(n))$ будет та функция $c*g(n)$, для которой найдется константа c_1 и объем данных N, при которых $c_1*g(n) \geq f(n)$, где $n \geq N$. Это условие будет соблюдено, если предел вида (4.3) будет больше или равен единице при больших n.

$$\lim_{n \to 10^9} \frac{g(n)}{f(n)} \geq 1.0 \quad (4.3)$$

Объем данных n должен стремиться к большому числу (не менее 10^9), но не к бесконечности, так как из-за случайных выбросов в экспериментальных данных регрессионная модель могла быть построена не точно и в бесконечности её значение может оказаться выше необходимой g(n) (предел меньше 1). f(n) здесь не все уравнение регрессионной модели, а только старший его член, взятый с коэффициентом 1. Следует проводить проверку, выбирая очередное g(n) в представленном выше порядке и первая, удовлетворяющее условию функция в таком случае будет точной асимптотической оценкой $f(n) = O(g(n))$. Если начать проверку с больших g(n), то первая взятая функция будет удовлетворять условию, но в этом случае полученная асимптотическая оценка будет приблизительной.

Порядок выполнения работы

1. Откройте выбранную IDE и создайте проект на основе существующих программных кодов, реализующих алгоритм пирамидальной сортировки.
2. Подключите к проекту библиотеку JUnit.

3. Создайте юнит-тест на проверку правильности работы программы сортировки в соответствии с ранее описанными требованиями.

4. Оцените результаты, полученные в предыдущем пункте. Сделайте соответствующие выводы.

5. Создайте юнит-тест на проверку теоретической трудоёмкости программы сортировки в соответствии с ранее описанными требованиями.

6. Произведите замеры времени не менее 5 раз для каждого из наборов исходных данных от 1 до 10000 элементов.

7. Вычислите медиану по всем экспериментам для каждого значения объёма исходных данных.

8. Проанализируйте отфильтрованные данные и найдите наиболее подходящую им регрессионную модель (линию тренда).

9. Подберите асимптотическую оценку этой модели.

10. Сравните экспериментальную трудоёмкость с теоретической. Сделайте соответствующие выводы.

Содержание отчёта

1. Юнит-тест на проверку правильности работы программы. Необходимо предоставить его описание, программный код, результаты работы и выводы по ним.

2. Юнит-тест на проверку теоретической трудоёмкости программы. Необходимо предоставить его описание, программный код, результаты работы и их обработки с соответствующими графиками и пояснениями, а также выводы по данному этапу тестирования.

3. Общие вводы по лабораторной работе.

Лабораторная работа № 7-8
Тестирование безопасности

***Цель работы*:** ознакомиться с проблемами реализации политик безопасности в компьютерных системах на примере дискреционной модели.

Теоретические сведения

Под политикой безопасности понимают набор норм, правил и практических приемов, регулирующих управление, защиту и распределение ценной информации. Политика безопасности задает механизмы управления доступа к объекту, определяет как разрешенные, так и запрещенные доступы.

Политика безопасности реализуется посредством административно-организационных мер, физических и программно-технических средств и определяет архитектуру системы защиты. Для конкретной организации политика безопасности должна носить индивидуальный характер и зависеть от конкретной технологии обработки информации и используемых программных и технических средств.

Политика безопасности определяется способом управления доступом, который задаёт порядок доступа к объектам системы. Различают два основных вида политики безопасности: избирательную и полномочную.

Избирательная политика безопасности основана на избирательном способе управления доступом. Избирательное (или дискреционное) управление доступом характеризуется заданным администратором множеством разрешенных отношений доступа (например, в виде троек объект – субъект – тип доступа). Обычно для описания свойств избирательного управления доступом применяют математическую модель на основе матрицы доступа.

Матрица доступа представляет собой матрицу, в которой столбец соответствует объекту системы, а строка – субъекту. На пересечении столбца и строки матрицы указывается тип разрешенного доступа субъекта к объекту. Обычно выделяют такие типы доступа субъекта к объекту, как «доступ на чтение», «доступ на запись», «доступ на исполнение» и т.п. Матрица доступа является самым простым подходом к моделированию систем управления доступом. Однако она служит основой для сложных моделей, более адекватно описывающих реальные автоматизированные системы обработки информации (АСОИ).

Избирательная политика безопасности широко применяется в АСОИ коммерческого сектора, так как её реализация соответствует требованиям коммерческих организаций по разграничению доступа и подотчетности, а также имеет приемлемую стоимость.

Полномочная политика безопасности основана на полномочном (мандатном) способе управления доступом. Полномочное (или мандатное) управление доступом характеризуется совокупностью правил предоставления доступа, определенных на множестве атрибутов безопасности субъектов и объектов, например, в зависимости от метки

конфиденциальности информации и уровня допуска пользователя. Полномочное управление доступом подразумевает, что:

1) все субъекты и объекты системы однозначно идентифицированы;

2) каждому объекту системы присвоена метка конфиденциальности информации, определяющая ценность содержащейся в нем информации;

3) каждому субъекту системы присвоен определенный уровень допуска, определяющий максимальное значение метки конфиденциальности информации объектов, к которым субъект имеет доступ.

Чем важнее объект, тем выше его метка конфиденциальности. Поэтому наиболее защищенными оказываются объекты с наиболее высокими значениями метки конфиденциальности.

Основное назначение полномочной политики безопасности – регулирование доступа субъектов системы к объектам с различными уровнями конфиденциальности, предотвращение утечки информации с верхних уровней должностной иерархии на нижние, а также блокирование возможных проникновений с нижних уровней на верхние.

При выборе и реализации политики безопасности в компьютерной системе, как правило, работают следующие шаги:

1. В информационную структуру вносится структура ценностей (определяется ценность информации) и проводится анализ угроз и рисков для информации и информационного обмена.

2. Определяются правила использования для любого информационного процесса, права доступа к элементам информации с учетом данной оценки ценностей.

Реализация политики безопасности должна быть четко продумана. Результатом ошибочного или бездумного определения правил политики безопасности, как правило, является разрушение ценности информации без нарушения политики.

Дискреционная политика безопасности

Пусть O – множество объектов, U – множество пользователей, S – множество действий пользователей над объектами. Тогда дискреционная политика определяет отображение $O \rightarrow U$ (объектов на пользователей-субъектов). В соответствии с данным отображением, каждый объект $O_j \in O$ объявляется собственностью соответствующего пользователя $U_k \in U$, который может выполнять над ними определенную совокупность действий $S_i \in S$, в которую могут входить несколько элементарных действий (чтение, запись, модификация и т.д.). Пользователь, являющийся собственником объекта, иногда имеет право передавать часть или все права другим пользователям (обладание администраторскими правами).

Указанные права доступа пользователей-субъектов к объектам компьютерной системы записываются в виде так называемой матрицы доступа. На пересечении i-й строки и j-ого столбца данной матрицы располагается элемент S_{ij} – множество разрешенных действий j-ого пользователя над i-м объектом.

Пример. Пусть имеем множество из трёх пользователей {Администратор, Гость, Пользователь_1} и множество из четырёх объектов {Файл_1, Файл_2, CD-RW, Дисковод}. Множество возможных действий включает следующие: {Чтение, Запись, Передача прав другому пользователю}. Действие «Полные права» разрешает выполнение всех трёх действий, действие «Запрет» запрещает выполнение всех перечисленных действий. В данном случае, матрица доступа, описывающая дискреционную политику безопасности, может выглядеть следующим образом.

Таблица 1. Пример матрицы доступа

Объект / Субъект	Файл_1	Файл_2	CD-RW	Дисковод
1. Администратор	Полные права	Полные права	Полные права	Полные права
2. Гость	Запрет	Чтение	Чтение	Запрет
3. Пользователь_1	Чтение, передача прав	Чтение, запись	Полные права	Запрет

Например, Пользователь_1 имеет права на чтение и запись в Файл_2. Передавать же свои права другому пользователю он не может.

Пользователь, обладающий правами передачи своих прав доступа к объекту другому пользователю, может сделать это. При этом, пользователь, передающий права, может указать непосредственно, какие из своих прав он передает другому.

Например, если Пользователь_1 передает право доступа к Файлу_1 на чтение пользователю Гость, то у пользователя Гость появляется право чтения из Файла_1.

Задание на лабораторную работу

Пусть множество *S* возможных операций над объектами компьютерной системы задано следующим образом: *S* = {«Доступ на чтение», «Доступ на запись», «Передача прав»}.

1. Получить данные о количестве пользователей и объектов компьютерной системы из табл. 2, соответственно варианту.

2. Реализовать программный модуль, создающий матрицу доступа пользователей к объектам компьютерной системы. Реализация данного модуля подразумевает следующее:

2.1. Необходимо выбрать идентификаторы пользователей, которые будут использоваться при их входе в компьютерную систему (по одному идентификатору для каждого пользователя, количество пользователей указано для варианта). Например, множество из трёх идентификаторов пользователей {Ivan, Sergey, Boris}. Один из данных идентификаторов должен соответствовать администратору компьютерной системы (пользователю, обладающему полными правами доступа ко всем объектам).

2.2. Реализовать программное заполнение матрицы доступа, содержащей количество пользователей и объектов, соответственно Вашему варианту.

2.2.1. При заполнении матрицы доступа необходимо учитывать, что один из пользователей должен являться администратором системы (допустим, Ivan). Для него права доступа ко всем объектам должны быть выставлены как полные.

2.2.2. Права остальных пользователей для доступа к объектам компьютерной системы должны заполняться случайным образом с помощью датчика случайных чисел. При заполнении матрицы доступа необходимо учитывать, что пользователь может иметь несколько прав доступа к некоторому объекту компьютерной системы, иметь полные права, либо совсем не иметь прав.

2.2.3. Реализовать программный модуль, демонстрирующий работу в дискреционной модели политики безопасности.

3. Данный модуль должен выполнять следующие функции:

3.1. При запуске модуля должен запрашиваться идентификатор пользователя (проводится идентификация пользователя), при успешной идентификации пользователя должен осуществляться вход в систему, при неуспешной – выводиться соответствующее сообщение.

3.2. При входе в систему после успешной идентификации пользователя на экране должен распечатываться список всех объектов системы с указанием перечня всех доступных прав доступа идентифицированного пользователя к данным объектам. Вывод можно осуществить, например, следующим образом:

User: Boris
Идентификация прошла успешно, добро пожаловать в систему
Перечень Ваших прав:
Объект1: Чтение
Объект2: Запрет
Объект3: Чтение, Запись
Объект4: Полные права
Жду ваших указаний >

3.3. После вывода на экран перечня прав доступа пользователя к объектам компьютерной системы, необходимо организовать ожидание указаний пользователя на осуществление действий над объектами в компьютерной системе. После получения команды от пользователя, на экран необходимо вывести сообщение об успешности либо не успешности операции. При выполнении операции передачи прав (grant) должна модифицироваться матрица доступа. Программа должна поддерживать операцию выхода из системы (quit), после которой запрашивается идентификатор пользователя. Диалог можно организовать, например, так:

Жду ваших указаний > read
Над каким объектом производится операция? 1
Операция прошла успешно
Жду ваших указаний > write
Над каким объектом производится операция? 2
Отказ в выполнении операции. У Вас нет прав для ее осуществления

Жду ваших указаний > grant
Право на какой объект передается? 3
Отказ в выполнении операции. У Вас нет прав для ее осуществления
Жду ваших указаний > grant
Право на какой объект передается? 4
Какое право передается? read
Какому пользователю передается право? Ivan
Операция прошла успешно
Жду ваших указаний > quit
Работа пользователя Boris завершена. До свидания.
User:

4. Выполнить тестирование разработанной программы, продемонстрировав реализованную модель дискреционной политики безопасности.

5. Оформить отчет по лабораторной работе.

Таблица 2. Варианты заданий

Вариант	Количество субъектов доступа (пользователей)	Количество объектов доступа
1	3	3
2	4	4
3	5	4
4	6	5
5	7	6
6	8	3
7	9	4
8	10	4
9	3	5
10	4	6
11	5	3
12	6	4
13	7	4
14	8	5
15	9	6

Контрольные вопросы

1. Что понимается под политикой безопасности в компьютерной системе?

2. В чем заключается модель дискреционной политики безопасности в компьютерной системе?

3. Что понимается под матрицей доступа в дискреционной политике безопасности? Что хранится в данной матрице?

4. Какие действия производятся над матрицей доступа в том случае, когда один субъект передает другому субъекту свои права доступа к объекту компьюерной системы?

Лабораторная работа №9.
Нагрузочное тестирование, стрессовое тестирование.

Цель: С помощью систем нагрузочного тестирования определить производительность web-серверов Apache и Nginx, добиться отказа в обслуживании.

Термины и определения

Нагрузочное тестирование - это автоматизированный процесс, имитирующий одновременную работу определенного количества пользователей на каком-либо общем ресурсе.

Приложение - тестируемое прикладное программное обеспечение.

Виртуальный пользователь - программный процесс, который циклически выполняет моделируемые операции.

Итерация - один повтор в цикле операции.

Интенсивность выполнения операции - частота выполнения операций в единицу времени, в тестовых скриптах задается интервалом времени между итерациями.

Нагрузка - совокупное количество попыток выполнить операции на общем ресурсе. Создается или пользовательской (клиентской) активностью или нагрузочными скриптами.

Производительность - количество выполняемых приложением операций в единицу времени.

Масштабируемость приложения - пропорциональный рост производительности при увеличении нагрузки.

Профилем нагрузки называется набор операций с заданными интенсивностями, полученными на основе статистики.

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

Тест производительности, бенчмарк (англ. benchmark) — контрольная задача, необходимая для определения сравнительных характеристик производительности компьютерной системы.

Успешное прохождение ряда тестов является свидетельством стабильности системы в штатном и в разогнанном режимах.

Цели нагрузочного тестирования

1. Оценка работоспособности и производительности приложения на этапе разработки и при передаче в эксплуатацию
2. Оценка работоспособности и производительности приложения на этапе выпуска новых релизов, патч-сетов
3. Оптимизация производительности приложения, включая оптимизацию программного кода и настройку серверов
4. Подбор соответствующей для данного приложения аппаратной и программной платформы, а также нужной конфигурации сервера

Виды нагрузочного теста

Нагрузочный (Load-testing) – определяет работоспособность системы при некотором строго заданном уровне нагрузки (планируемой, рабочей).

Устойчивости (Stress) – используется для проверки параметров системы в экстремальных условиях и условиях сверхнагрузки, основная задача во время испытания - нарушить нормальную работу системы. Позволяет определить минимальные системные требования для работы приложения, оценить предельные возможности системы и факторы, которые ограничивают эти возможности. В рамках теста также определяется возможность системы сохранить целостность данных при возникновении внештатных аварийных ситуаций.

Производительности (Performance) – комплексный тест, включает в себя предыдущие два режима тестирования и предназначен для общей оценки всех показателей системы.

Результат теста – представляет собой максимально возможное число пользователей, которые могут единовременно получить доступ к веб-ресурсу, число запросов, которое в состоянии обработать приложение, или время ответа сервера. На основе этой информации, веб-мастер и сетевой администратор смогут заранее выявить слабые места, возникающие из-за несбалансированной работы компонентов (базы данных, маршрутизаторы, кэширующий и прокси-сервер, брандмауэры и др.), и исправить ситуацию, прежде чем система будет запущена в рабочем режиме.

Использование Apache benchmark tool

Apache benchmark — одна из самых простых утилит, которая применяется для нагрузочного тестирования сайта. Идет в комплекте с веб сервером Apache, в первоначальной настройке не нуждается. Задача, которая ставится перед Apache benchmark — показать, какое количество запросов сможет выдержать веб сервер и как быстро он их обработает.

Пример нагрузки на сервер в 5000 последовательных запросов:

<ab -n 5000 http://192.168.1.116/index.html>

Пример нагрузки на сервер в 5000 запросов, но 500 из них будут направлены на сервер одновременно (параллельные запросы):

<ab -n 5000 -c 500 http://192.168.1.116/index.html>

Примечание

Для выполнения лабораторной работы меняйте значения после -n и -c, чтобы узнать с каким количеством запросов может справиться сервер. На этих примерах выполнены HTML-запросы, для тестирования на PHP-запросы измените цель на index.php (В лабораторной работе №8 есть информация о PHP).

Использование httperf

Еще одно консольное приложение, используемое также для создания нужного количества параллельных запросов - httperf.

Его отличие от ab в том, что httperf посылает запросы согласно своим настройкам, невзирая на то, отвечает сервер на них или уже нет. Таким образом можно определить не только какую максимальную нагрузку может

выдержать сервер, но и как будет себя вести сервер в момент, когда нагрузка достигла своего пика

Пример запуска 100 запросов от 10 посетителей параллельно:

```
httperf --port 80 --server <domain> --uri=/ --num-conns=100 --rate=10
```

Использование Siege

Установка:

```
sudo apt-get install siege
```

Количество запросов не лимитируется, но можно задавать время в течение которого выполнять тестирование. Пример: 5 пользователей, которые безостановочно загружают главную страницу в течении одной минуты.

```
siege -c 5 -b -t 1m ip-адрес
```

Балансировщик нагрузки Nginx

Балансировка нагрузки (англ. load balancing) — метод распределения заданий между несколькими сетевыми устройствами (например, серверами) с целью оптимизации использования ресурсов, сокращения времени обслуживания запросов, горизонтального масштабирования кластера (динамическое добавление/удаление устройств), а также обеспечения отказоустойчивости (резервирования).

Установка:

```
sudo apt-get install nginx
```

Настройка:

```
sudo nano /etc/nginx/sites-available/default

upstream web_backend {
    server 192.168.1.113;
    server 192.168.1.114;
}
server {
    listen 80;
    location / {
        proxy_set_header XForwardedFor $proxy_add_x_forwarded_for;
        proxy_pass http://web_backend;
    }
}
```

После настройки перезапускаем Nginx

```
sudo service nginx reload
```

Методы балансировки нагрузки (описываются в начале секции upstream):

ip_hash - согласно этому методу запросы от одного и того же клиента будут всегда отправляться на один и тот же backend сервер на основе информации об ip адресе клиента. Не совместим с параметром weight.

least_conn - запросы будут отправляться на сервер с наименьшим количеством активных соединений.

round-robin - режим по умолчанию. То есть если вы не задали ни один из вышеупомянутых способов балансировки - запросы будут доставляться по очереди на все сервера в равной степени.

Задания к лабораторной работе

Нагрузочное тестирование веб-сервера с Apache

Для тестирования используются 2 машины – одна с установленным и работающим Apache, вторая будет отсылать запросы и делать выводы о производительности web-сервера.

Тестирование на PHP-запросы:

- Определить максимальное число параллельных запросов, при котором сервер нас не будет блокировать.
- Провести тест при использовании максимального числа запросов.

Тестирование на HTML-запросы:

- Определить максимальное число параллельных запросов
- Провести тест при использовании максимального числа запросов.

Провести сравнение результатов и сформировать выводы.

Нагрузочное тестирование веб-сервера с Nginx.

Для тестирования используется 2 виртуальные машины – одна с установленным и работающим Nginx, которой будут отсылаться запросы, другая будет отсылать эти самые запросы и делать выводы о производительности веб-сервера с Nginx.

Примечание

<sudo apt-get nginx> - Установка Nginx

Тестирование на PHP-запросы:

- Определить максимальное число параллельных запросов, при котором сервер нас не будет блокировать.
- Провести тест при использовании максимального числа запросов.
- Сравнить с результатами, полученными при тестировании Apache

Тестирование на HTML-запросы:

- Определить максимальное число параллельных запросов.
- Провести тест при использовании максимального числа запросов.
- Сравнить с результатами, полученными при тестировании Apache

Провести сравнение результатов и сформировать выводы.

Нагрузочное тестирование веб-серверов Apache с балансировщиком нагрузки.

Для тестирования используется 4 машины – две одинаковые с установленным и работающим Apache в качестве веб-серверов, которые соединены с третьей машиной, которая выполняет роль балансировщика нагрузки, на нем работает Nginx, четвертая машина будет отсылать эти запросы серверу и делать выводы о производительности данной связки из балансировщика нагрузки на Nginx и двумя веб-серверами Apache.

Тестирование на PHP-запросы:

- Провести тест при использовании максимального для Apache числа запросов
- Провести тест при использовании максимального для Nginx числа запросов
- Сравнить с предыдущими результатами и сформировать выводы

Тестирование на HTML-запросы:

- Провести тест при использовании максимального для Apache числа запросов

- Провести тест при использовании максимального для Nginx числа запросов
- Сравнить с предыдущими результатами и сформировать выводы

Нагрузочное тестирование веб-серверов Nginx с балансировщиком нагрузки.

Для тестирования используется 4 виртуальные машины – две одинаковые с установленным и работающим Nginx в качестве веб- серверов, которые соединены с третьей машиной, которая выполняет роль балансировщика нагрузки, на нем работает Nginx, четвертая машина будет отсылать эти запросы серверу и делать выводы о производительности данной связки из балансировщика нагрузки на Nginx и двумя веб-серверами Nginx.

Тестирование на PHP-запросы:

- Провести тест при использовании максимального для Apache числа запросов
- Провести тест при использовании максимального для Nginx числа запросов
- Сравнить с предыдущими результатами и сформировать выводы

Тестирование на HTML-запросы:

- Провести тест при использовании максимального для Apache числа запросов
- Провести тест при использовании максимального для Nginx числа запросов
- Сравнить с предыдущими результатами и сформировать выводы

Итоговая таблица сравнения

		Максимальное число запросов	**Запросы/сек**	**Время, затрачиваемое на запрос, мс**	**% успешных запросов**
Apache	PHP				
	HTML				
LB + Apache	PHP				
	HTML				
Nginx	PHP				
	HTML				
LB + Nginx	PHP				
	HTML				

Вопросы к лабораторной работе

1. Назначение нагрузочного тестирования?
2. Что такое нагрузка?
3. Как указать ab сделать нагрузку в 10000 запросов, 500 из которых будут направлены одновременно? Перестанет ли ваш сервер принимать входящие подключения?

4. Можно ли протестировать при помощи ab, httperf и Siege другие web-сервера? Назовите примеры.
5. Влияет ли использование скриптовых языков программирования (например, PHP) на производительность web-сервера? Объясните почему.
6. Для чего нужен балансировщик нагрузки?
7. Какие существуют методы балансировки нагрузки в nginx?

Лабораторная работа № 10-11
Модульное тестирование

Цель работы: Овладение навыками модульного тестирования.

Общие сведения

Модульное тестирование - это тестирование программы на уровне отдельно взятых модулей, функций или классов. Цель модульного тестирования состоит в выявлении локализованных в модуле ошибок в реализации алгоритмов, а также в определении степени готовности системы к переходу на следующий уровень разработки и тестирования. Модульное тестирование проводится по принципу "белого ящика", то есть основывается на знании внутренней структуры программы, и часто включает те или иные методы анализа покрытия кода.

Модульное тестирование обычно подразумевает создание вокруг каждого модуля определенной среды, включающей заглушки для всех интерфейсов тестируемого модуля. Некоторые из них могут использоваться для подачи входных значений, другие для анализа результатов, присутствие третьих может быть продиктовано

требованиями, накладываемыми компилятором и сборщиком. На уровне модульного тестирования проще всего обнаружить дефекты, связанные с алгоритмическими ошибками и ошибками кодирования алгоритмов, типа работы с условиями и счетчиками циклов, а также с использованием локальных переменных и ресурсов.

Ошибки, связанные с неверной трактовкой данных, некорректной реализацией интерфейсов, совместимостью, производительностью и т.п. обычно пропускаются на уровне модульного тестирования и выявляются на более поздних стадиях тестирования.

Задание

1) "Мозговая атака":

Этапы:

- Получить вопрос (задание) для обсуждения.
- Задать вопросы относительно не понятных моментах в вопросе (задании).
- Высказать свои мысли по данному вопросу (заданию).
- Записать все прозвучавшие высказывания с уточнениями.
- По окончанию прочитать все, что было записано.
- Обсудить все варианты ответов.
- Выяснить, как можно использовать полученные результаты при выполнении данной лабораторной работы.

2) "Работа в команде"

Этапы:

- Разбиться на команды.
- Реализовать полученный вопрос (задание), согласно

технологии TDD.
- Представить результаты

Лабораторная работа № 12-13
Тестирование интеграций

Цель работы: Овладение навыками интеграционного тестирования

Общие сведения

Интеграционное тестирование называют еще тестированием архитектуры системы. С одной стороны, это название обусловлено тем, что интеграционные тесты включают в себя проверки всех возможных видов взаимодействий между программными модулями и элементами, которые определяются в архитектуре системы - таким образом, интеграционные тесты проверяют полноту взаимодействий в тестируемой реализации системы. С другой стороны, результаты выполнения интеграционных тестов - один из основных источников информации для процесса улучшения и уточнения архитектуры системы, межмодульных и межкомпонентных интерфейсов. Т.е., с этой точки зрения, интеграционные тесты проверяют корректность взаимодействия компонент системы.

В результате проведения интеграционного тестирования и устранения всех выявленных дефектов получается согласованная и целостная архитектура программной системы, т.е. можно считать, что интеграционное тестирование - это тестирование архитектуры и низкоуровневых функциональных требований.

Интеграционное тестирование, как правило, представляет собой итеративный процесс, при котором проверяется совокупность модулей, возрастающая от итерации к итерации. В интеграционном тестирование выделяют три методы выполнения: восходящее тестирование; монолитное тестирование; нисходящее тестирование.

Задание

Согласно варианту провести один из методов интеграционного тестирования.

Лабораторная работа № 14-15
Системное тестирование

Цель работы: Овладение навыками системного тестирования

Общие сведения

Системное тестирование - один из самых сложных видов тестирования. На этапе системного тестирования проводится не только функциональное тестирование, но и оценка характеристик качества системы - ее устойчивости, надежности, безопасности и производительности. На этом этапе выявляются многие проблемы внешних интерфейсов системы, связанные с неверным взаимодействием с другими системами, аппаратным обеспечением, неверным распределением памяти, отсутствием корректного освобождения ресурсов и т.п.

После завершения системного тестирования разработка переходит в фазу приемо-сдаточных испытаний (для программных систем, разрабатываемых на заказ) или в фазу альфа- и бета-тестирования (для программных систем общего применения).

Системное тестирование проводится в несколько фаз, на каждой из которых проверяется один из аспектов поведения системы, т.е. проводится один из типов системного тестирования. Все эти фазы могут протекать одновременно или последовательно. Следующий раздел посвящен рассмотрению особенностей каждого из типов системного тестирования на каждой фазе.

Виды системного тестирования:

1)функциональное тестирование;

2)тестирование производительности;

3)нагрузочное или стрессовое тестирование;

4)тестирование конфигурации;

5)тестирование безопасности;

6)тестирование надежности и восстановления после сбоев;

7)тестирование удобства использования.

Задание

Согласно варианту провести несколько видов системного тестирования.

Лабораторная работа 16-17

«Конфигурационное тестирование»

Теоретические сведения

Конфигурационное тестирование или Configuration Testing

Конфигурационное тестирование (Configuration Testing) — специальный вид тестирования, направленный на проверку работы программного обеспечения при различных конфигурациях системы (заявленных платформах, поддерживаемых драйверах, при различных конфигурациях компьютеров и т.д.)

В зависимости от типа проекта конфигурационное тестирование может иметь разные цели:

1. Проект по профилированию работы системы **Цель Тестирования**: определить оптимальную конфигурацию оборудования, обеспечивающую требуемые характеристики производительности и времени реакции тестируемой системы.
2. Проект по миграции системы с одной платформы на другую **Цель Тестирования**: Проверить объект тестирования на совместимость с объявленным в спецификации оборудованием, операционными системами и программными продуктами третьих фирм.

Примечание: В **ISTQB Syllabus** вообще не говорится о таком виде тестирования как конфигурационное. Согласно глоссарию, данный вид тестирования рассматривается там как тестирование портируемости: configuration testing: See **portability testing**. **portability testing**: The process of testing to determine the portability of a software product.

Уровни проведения тестирования

Для клиент-серверных приложений конфигурационное тестирование можно условно разделить на два уровня (для некоторых типов приложений может быть актуален только один):

1. Серверный
2. Клиентский

На первом (серверном) уровне, тестируется взаимодействие выпускаемого программного обеспечения с окружением, в которое оно будет установлено:

1. Аппаратные средства (тип и количество процессоров, объем памяти, характеристики сети / сетевых адаптеров и т.д.)
2. Программные средства (ОС, драйвера и библиотеки, стороннее ПО, влияющее на работу приложения и т.д.)

Основной упор здесь делается на тестирование с целью определения оптимальной конфигурации оборудования, удовлетворяющего требуемым характеристикам качества (эффективность, портативность, удобство сопровождения, надежность).

На следующем (клиентском) уровне, программное обеспечение тестируется с позиции его конечного пользователя и конфигурации его

рабочей станции. На этом этапе будут протестированы следующие характеристики: удобство использования, функциональность. Для этого необходимо будет провести ряд тестов с различными конфигурациями рабочих станций:

1. Тип, версия и битность операционной системы (подобный вид тестирования называется **кросс-платформенное тестирование**)
2. Тип и версия Web барузера, в случае если тестируется Web приложение (подобный вид тестирования называется **кросс-браузерное тестирование**)
3. Тип и модель видео адаптера (при тестировании игр это очень важно)
4. Работа приложения при различных разрешениях экрана
5. Версии драйверов, библиотек и т.д. (для JAVA приложений версия JAVA машины очень важна, тоже можно сказать и для .NET приложений касательно версии .NET библиотеки)

и т.д.

Порядок проведения тестирования

Перед началом проведения конфигурационного тестирования рекомендуется:

- создавать матрицу покрытия (**матрица покрытия** - это таблица, в которую заносят все возможные конфигурации),
- проводить приоритезацию конфигураций (на практике, скорее всего, все желаемые конфигурации проверить не получится),
- шаг за шагом, в соответствии с расставленными приоритетами, проверяют каждую конфигурацию.

Уже на начальном этапе становится очевидно, что чем больше требований к работе приложения при различных конфигурациях рабочих станций, тем больше тестов нам необходимо будет провести. В связи с этим, рекомендуем, по возможности, автоматизировать этот процесс, так как именно при конфигурационном тестировании автоматизация реально помогает сэкономить время и ресурсы. Конечно же автоматизированное тестирование не является панацеей, но в данном случае оно окажется очень эффективным помощником.

Обобщим:

- конфигурационным называется тестирование совместимости выпускаемого продукта (программное обеспечение) с различным аппаратным и программным средствами
- основные цели - определение оптимальной конфигурации и проверка совместимости приложения с требуемым окружением (оборудованием, ОС и т.д.)
- автоматизация конфигурационного тестирования позволяет избежать лишних расходов

Лабораторная работа №19

Методы автоматизации исполнения тестов.

Цель: изучить подробно средства автоматизации тестов, на примере IBM Rational Functional Tester, IBM Rational Performance Tester, TestComplete10.

Краткая теория и ход выполнения работы

IBM Rational Functional Tester

IBM Rational Functional Tester - это инструмент автоматического функционального тестирования и регрессионного тестирования. Это программное обеспечение предоставляет функции автоматического тестирования для функционального, регрессионного тестирования, ориентированного на данные. Rational Function Tester поддерживает ряд приложений, таких как веб-приложения, приложения для .Net, Java, Siebel, SAP, приложения на основе эмулятора терминала, PowerBuilder, Ajax, Adobe Flex, Dojo Toolkit,

GEF, документы Adobe PDF, zSeries, iSeries и pSeries.

IBM Rational Performance Tester

Performance Tester - это инструмент тестирования производительности, с помощью которого можно выявлять проблемы системной производительности и их причины. Rational Performance Tester предоставляет тестировщикам средства автоматизированного тестирования, позволяющие выполнять функциональное тестирование, регрессивное тестирование, тестирование пользовательского интерфейса и тестирование, управляемое данными.

IBM Rational Quality Manager

Rational Quality Manager — это решение для совместной работы, предназначенное для обеспечения высокого качества программного обеспечения и систем с учетом особенностей бизнеса и поддерживающее практически все платформы и типы тестирования. Это программный продукт обеспечивает удобное совместное использование данных в рабочих группах, поддерживает автоматизацию для сокращения сроков выполнения проектов и позволяет создавать отчеты о ходе работы для принятия более обоснованных решений.

IBM Rational Test RealTime

IBM Rational Test RealTime - межплатформное решение, обеспечивающее тестирование компонентов и анализ их работы. Создано специально для разработчиков сложных систем встроенных приложений, приложений реального времени и других типов межплатформных программных продуктов. Это ПО позволяет отлаживать и исправлять ошибки до того, как они попадут в программный код готового продукта.

TestComplete 10

TestComplete — это инструмент для автоматизации тестирования, позволяющий Вам создавать и выполнять тесты для Windows, .NET, WPF, Visual C++, Visual Basic, Delphi, C++ Builder, Java и веб-приложений, как простых, так и обладающих богатой функциональностью. TestComplete ориентирован как на функциональное так и модульное тестирование. Используя TestComplete, Вы с легкостью сможете создавать и

автоматизировать тесты для ваших приложений. Новые возможности TestComplete 10: • поддержка Microsoft Visual Studio 2013 и Embarcadero RAD Studio XE7 • поддержка Windows 8.1 • Поддержка Ready! API и SoapUI.

TestExecute

TestExecute - это утилита, предназначенная для выполнения TestComplete проектов на компьютерах, где не установлен TestComplete. TestExecute дает тестировщикам возможность демонстрировать работу тестов в реальных условиях – например, на машине клиента с учетом особенностей его программного окружения – и освобождает от необходимости устанавливать TestComplete на целевые компьютеры. TestExecute удобно использовать для распределенного тестирования.

Borland SilkTest

SilkTest инструмент функционального тестирования приложений. Дает тестировщикам программного обеспечения возможность легко успевать за темпами разработки. Silk Test разработан в расчете на интеграцию со средой разработки, он обеспечивает постоянные и воспроизводимые результаты тестов в условиях популярных сегодня укороченных жизненных циклов выпуска продукта. Silk Test поддерживает гибкие циклы разработки, помогая тщательно проверять надежность приложений при помощи сложных тестов.

IBM Rational Application Performance Analyzer

IBM Rational Application Performance Analyzer позволяет пользователям выявлять программный код приложения, который является причиной низкой производительности.

IBM Rational Performance Test Pack Virtual Testers

IBM Rational Performance Tester - это инструмент тестирования производительности, с помощью которого можно выявлять проблемы системной производительности и их причины. IBM Rational Performance Tester предоставляет тестировщикам средства автоматизированного тестирования, позволяющие выполнять функциональное тестирование, регрессивное тестирование, тестирование пользовательского интерфейса и тестирование, управляемое данными.

IBM Rational Performance Test Server

IBM Rational Performance Test Server — это средство тестирования рабочей нагрузки, которое позволяет проверить производительность и масштабируемость приложения. Решение IBM Rational Performance Test Server максимизирует инфраструктуру тестирования для быстрого развертывания сценариев загрузки и проведения масштабного тестирования системы.

IBM Rational Performance Tester Extension for Siebel Test Automation

Rational Performance Tester Extension for Siebel Test Automation это быстрый поиск и устранение неисправностей за счет проверки типов данных Siebel. IBM Rational Performance Tester Extension for Siebel Test Automation расширяет возможности IBM Rational Performance Tester за счет предоставления средств тестирования производительности и работы при различных нагрузках для приложений Siebel 7.7. Сочетая удобные в использовании функции с глубокой детализацией, Rational Performance Tester Extension for Siebel Test Automation упрощает разработку тестов, моделирование нагрузки и процессы сбора данных, что позволяет полностью проверить готовность приложений к продуктивному использованию.

IBM Rational Performance Tester for z/OS

IBM Rational Performance Tester for z/OS - это единственное средство для тестирования производительности под нагрузкой, позволяющее использовать оборудование серии zSeries для моделирования нагрузки. Rational Performance Tester for z/OS сочетает в себе множество удобных функций с широкими возможностями для тестирования, которые упрощают создание тестов, моделирование нагрузки и анализ. Это помогает разработчикам создавать приложения, способные справляться с требуемой нагрузкой.

IBM RATIONAL Service Tester For Soa Quality

IBM Rational Service Tester for SOA Quality - это инструмент обеспечения качества SOA-приложений, построенных на основе Web-служб. Предоставляет тестировщикам возможности автоматического тестирования без сценариев, обеспечивающие выполнение функционального тестирования, регрессивного тестирования и тестирования производительности Web-служб без пользовательского интерфейса.

IBM Rational Test Virtualization Server

IBM Rational Test Virtualization Server решение, помогающие оптимизировать качество проекта с помощью центра объединенного управления тестированием, предоставляющего интегрированную поддержку жизненного цикла практически для любой платформы и типа тестирования. IBM Rational Test Virtualization Server позволяет создавать, изменять и разворачивать приложения для упрощенного, эффективного тестирования. Это решение ускоряет доставку сложных сред тестирования и позволяет выполнять тестирование интеграции раньше и более часто в цикле разработки.

IBM Rational Test Workbench

IBM Rational Test Workbench средство для комплексного тестирования функций, регрессии, загрузки и интеграции. IBM Rational Test Workbench создавает интеллектуальные и взаимосвязанные приложения предприятий, которые могут быть развернуты в традиционных и облачных инфраструктурах. IBM Rational Test Workbench сокращает время циклов

тестирования и раньше переносить тестирование интеграции в жизненном цикле разработки.

SmartBear Collaborator (ранее: CodeCollaborator Code)

SmartBear Collaborator – это лучший коммерческий инструмент для критического анализа кода разрабатываемых программ. Также Collaborator предоставляет возможность анализировать изображения, проектную документацию, документы Microsoft Office гораздо быстрее, чем обычно. Критический анализ кода - это лучший способ повысить качество разрабатываемого программного обеспечения. Благодаря Collaborator проверка кода становится еще эффективнее. Collaborator упрощает и ускоряет процесс анализа кода, предоставляя подробные отчеты, что поможет улучшить Ваши программы.

AQtime Pro

AQtime - это инструмент для повышения производительности и улучшения качества приложений. AQtime может анализировать 32 и 64-ех разрядные Windows, .NET, Silverlight и Java приложения, созданные с помощью C#, VB.NET, Visual C++, Visual Basic, Delphi, C++Builder, Intel C++, Compaq Visual Fortran и GNU C++ компиляторов. AQtime также поддерживает работу с JScript и VBScript кодом. AQtime интегрируется в Visual Studio, а также в Embarcadero RAD Studio, что позволяет находить узкие места и оптимизировать ваши программы, не покидая среды разработки.

Задание для самостоятельной работы

1. Изучить теоретически существующие средства автоматизации тестов.
2. Законспектировать кратко теорию по лабораторной работе.
3. Ознакомится самостоятельно на практике со следующими средствами автоматизации тестов: IBM Rational Functional Tester, IBM Rational Quality Manager, TestComplete 10. Для этого воспользоваться ярлыками программ на рабочем столе ПК.
4. Оформить отчет и вывод о проделанной работе.

Отчет должен содержать:

1. № и тему лабораторной работы.
2. Выполненное задание по лабораторной работе.

Лабораторная работа №20-21
Автоматизация тестирования с помощью скриптов

Как создать свой тест?

В данном случае используется тот же принцип, что и при ручном тестировании:

- задать состояние окружения (StoreStat, AxlePar, RollerPar, StoreMessage, CommandStatus);
- ждать, когда в системе произойдет определенное событие;
- задать новое состояние окружения;
- и т.д.

Каждый тест должен быть представлен в следующем виде:

ЗАГОЛОВОКБЛОКWait <условие>БЛОКWait <условие>.....EndTest

Опишем эти элементы более подробно.

Описание заголовка

Заголовок теста состоит из:

- команды запуска сервера;
- команд global, устанавливающих доступ к глобальным *переменным состояния* окружения;
- команды StartTest, задающей имя теста (оно не обязательно должно совпадать с именем файла).

source bin\\srv.tcl // запуск серверaglobal StoreStat // статус складаglobal AxlePar // терминал осиglobal RollerPar // терминал подшипникаglobal CommandStatus // возвращаемое значение функции // SendStoreComglobal StoreMessage // сообщение от складаglobal rollers_found // 1 (можно подобрать подходящий // подшипник) или 0 (нельзя) global fds // строка для графы "Покрытие FDS" // в итоговой таблице результатов // тестирование (по умолчанию - Default)global last_command // последняя команда тестируемой системыglobal allowed // список разрешенных команд, их // получение должно вызывать ошибкуStartTest <имя> // запуск теста

Описание блока

Каждый блок устанавливает значения одной или более переменных окружения. Например:

Set StoreStat 32...

Описание Wait

Команда Wait используется для изменения состояния тестового окружения в ответ на действия системы.

При вызове процедуры Wait скрипт входит в состояние ожидания. Обращение со стороны системы приводит к завершению состояния ожидания. Если условие выполнено, то ответ на это обращение будет сформирован на основе следующего блока, иначе - на основе предыдущего.

Wait <команда> <время> <обязательно>

<команда> определяет команду системы, которая приведет к переходу в следующий блок. Можно использовать стандартные символы подстановки *, ?, [];

<время> определяет время ожидания. Если задать 0, то время не ограничено;

<обязательно> дает возможность не формировать ошибку, если за заданное время требуемая команда не была получена. Если задана 1, то в случае неполучения команды будет сформирована ошибка, а если 0, то ошибки не будет.

Примеры использования:

Wait ZZZZZZZ 10 0 - ждать 10 секунд (команда ZZZZZZZ является недопустимой и не может быть получена)

Wait "GetStoreStat *" 0 1 - ждать команды GetStoreStat с любыми параметрами неограниченное время

Wait "SendStoreCom 2 *" 10 1 - ждать команды SendStoreCom с первым параметром 2 и произвольными остальными параметрами в течение 10 секунд, если она не получена, будет зафиксирована ошибка

Wait ZZZZZZZ 0 0 - бесконечное ожидание.

Так как ожидание в процедуре Wait прерывается только при поступлении запроса от системы, то в случае отсутствия запросов (система зависла и ничего не делает) тест также останавливается. Для предотвращения такой ситуации используется следующая команда: Timeout <время>.

<время> определяет число секунд, после которого процесс тестирования будет прерван. При этом в журнал теста будет занесено сообщение TIMEOUT. Срабатывание таймаута приводит к завершению теста. Команда Timeout 0 отменяет ограничение времени.

Описание allowed

Рассмотрим переменную allowed более подробно. Она содержит список тех команд, получение которых разрешено и не должно вызывать ошибку. Например, в следующем примере вызовGetAxlePar не спровоцирует ошибку:

Set StoreStat 32Wait "GetStoreStat *" 0 1Wait "GetRollerPar" 0 1Set allowed [list "GetAxlePar"]Wait "SendStoreCom 1 0 0 1 0 0 0" 0 1

Однако ошибка все же возникнет, если данная команда в данной ситуации является недопустимой. Несмотря на то, что проверка реализована в скрипте srv.*tcl*, использование командыWait и списка allowed могут ужесточить проверку. Значение по умолчанию для allowed - " * ". В этом случае проверку проходят все команды. Вход в процедуру Wait вызывает расширение списка allowed до момента выхода из нее, так что ожидаемая команда системы всегда проходит проверку на принадлежность списку allowed. В списке allowed можно задействовать символы подстановки.

Как создать свой тест?

В данном случае под тестом мы будем понимать его представление в виде MSC-диаграммы. В качестве объектов мы будем рассматривать тест (Test) и тестируемую систему (Model). Обмен сообщениями между тестом и моделью показан на рис. 10.1.

Когда в системе происходит определенное событие, она запрашивает состояние окружения, посылая сигнал тестовому окружению.

Тест возвращает состояние окружения (StoreStat, AxlePar, RollerPar, StoreMessage, CommandStatus), посылая модели сигнал с параметрами в соответствии с запросом.

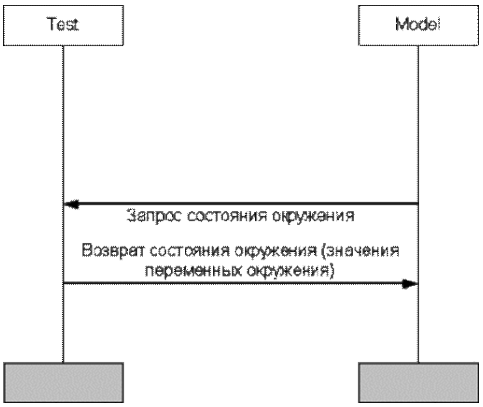
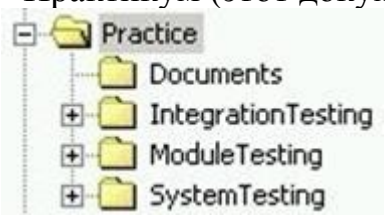


Рис. 10.1. Взаимодействие теста и модели

Структура и описание содержимого каталогов

В папке Documents находится:

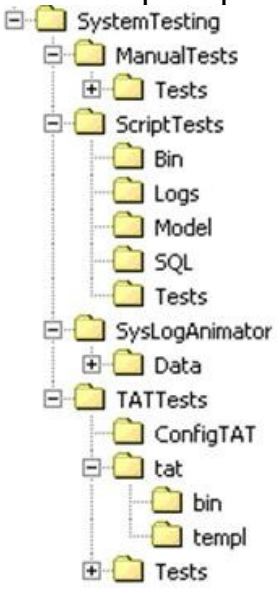
- *FDS*;
- HLD;
- Практикум (этот документ).



Папка IntegrationTesting содержит проект Visual Studio .NET с примером интеграционного теста.

Папка ModuleTesting содержит проект Visual Studio .NET с примером модульного теста.

В папке SystemTesting\ManualTests содержится проект Visual Studio .NET с примерами системных ручных тестов.



В папке SystemTesting\ScriptsTests содержатся примеры тестов с использованием скриптов:

\bin - содержит программу launcher.exe, файл srv.*tcl* и вспомогательные скрипты header.*tcl* и footer.*tcl*, используемые для формирования структуры html-отчета. Программаlauncher.exe запускает тестируемую систему и тестовый скрипт на выполнение, а после завершения теста завершает выполнение системы.

\logs - log-файлы пройденных тестов (*.log), log-файлы тестируемой системы (*.txt) и общий отчет summary.html.

\model - исполнимые файлы тестируемой системы.

\sql - SQL скрипты для установки заданного состояния базы данных.

\tests - тестовые скрипты на языке *TCL*.

Папка SystemTesting\TATTests содержит:

\ConfigTAT - программа ConfigTAT для настройки и запуска тестов *TAT*:

tat - система *автоматизации тестирования TAT*.

\Tests\config\config.xml - xml файл, описывающий тестируемую систему (AUT), тестовое окружение и сигналы между ними.

\Tests\Model - тестируемая система application under test.

\Tests*mpr* - *mpr*-файлы, описывающие тесты.

\Tests\SQLScripts - sql-скрипты для подготовки базы данных к конкретному тесту.

\Tests\Tests - *stencil* для Visio и MSC-диаграммы тестов в Visio.

\Tests\WareHouseTest№ - рабочие папки тестов.

\Tests\ *.tcf, tests.*ltc* - сохраненные конфигурации тестов и Testsuite- а для ConfigTAT.

Папка SysLogAnimator содержит основные файлы: библиотека *DirectX*, запускаемый файл LogAnimator.exe и конфигурационный файл LogAnimator.*cfg*.

Папка SysLogAnimator\Data содержит необходимые для работы аниматора графические файлы.

Описание MSC

Язык *диаграмм взаимодействия* (Message Sequence Charts, MSC) - это язык описания поведения системы в виде последовательности событий. События могут относиться к отдельным компонентам системы, к взаимодействиям между компонентами системы либо к взаимодействию между системой и ее окружением. Основное назначение диаграмм взаимодействия - описание последовательностей допустимых взаимодействий между компонентами системы и системой и ее окружением. Диаграммы изображаются в графическом виде, но существует также текстовая форма MSC-диаграмм. Обе формы переводятся взаимно однозначно друг в друга.

Разновидности *диаграмм взаимодействия* используются при разработке систем реального времени с 1960-х годов. Особое

распространение *диаграммы взаимодействия* получили в области разработки телекоммуникационных систем. Язык диаграмм взаимодействия стандартизован в 1992 году Международным Телекоммуникационным Союзом (ITU-T) (Рекомендация Z.120 1992). В настоящее время принята новая, значительно расширенная версия стандарта (Рекомендация Z.120 1996.).

Основные понятия

Диаграмма взаимодействия описывает последовательности событий, происходящих с набором объектов (системой взаимосвязанных компонентов). Дополнительно каждая система рассматривается как открытая, т.е. подразумевается наличие некоторого окружения системы, с которым система взаимодействует. Окружение также может задаваться в виде отдельного объекта.

Основным понятием диаграммы взаимодействий является трасса объекта. Для каждого объекта на диаграмме имеется отдельная вертикальная ось. На этой оси откладываются события, имеющие отношение к данному объекту. Считается, что все объекты существуют одновременно, и последовательности событий объектов развиваются параллельно. При описании объекта используются стартовый (прозрачный прямоугольник) и конечный (черный прямоугольник) символы объекта, обозначающие соответственно начало и конец описания объекта в данной MSC-диаграмме.

Взаимодействие между объектами (а также между объектом и окружением системы) осуществляется только при помощи обмена сообщениями (рис. 10.2).

Сообщение моделирует взаимодействие (т.е. обмен информацией) между двумя объектами системы или между объектом и окружением системы. С точки зрения системы, взаимодействие между двумя объектами разбивается на два сопряженных события: посылка сообщения одним объектом и прием сообщения другим объектом (рис. 10.2). Сообщения, приходящие из окружения системы, моделируются одним событием приема сообщения, а события, посылаемые в окружение, моделируются одним событием посылки сообщения. Сообщение имеет имя. Имя сообщения задает тип взаимодействия. Диаграмма может описывать несколько обменов сообщениями с одинаковым именем. Для уникальной идентификации конкретного обмена предусмотрен так называемый уникальный идентификатор обмена (message instance name), однако он используется только в текстовом представлении для снятия неоднозначности в описании сопряженных событий у различных объектов. В графическом представлении такой проблемы не возникает, так как сопряженные события представляются различными концами одного и того же графического объекта (стрелки от трассы одного объекта к трассе другого).

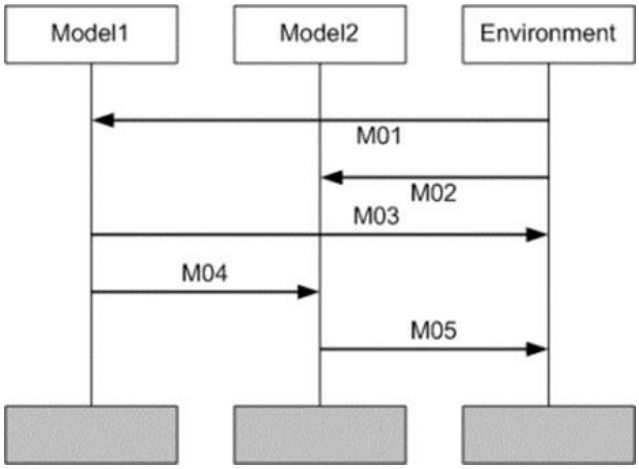


Рис. 10.2. Пример взаимодействия между объектами в MSC

Дополнительно, язык *диаграмм взаимодействия* позволяет описывать передачу информации в сообщении (рис. 10.3). С сообщением может быть *связан список* параметров. Каждый параметр моделирует передачу конкретной информации от одного объекта к другому. Язык диаграмм взаимодействия не определяет семантику параметров сообщения.

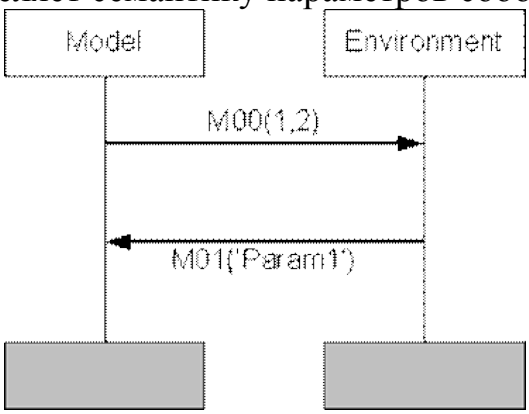


Рис. 10.3. Передача параметров сообщений

MSC-диаграммы позволяют создавать более сложные описания поведения системы с помощью специальных операторов. В MSC'96 используется четыре типа операторов: alt -*альтернативный оператор*, par - параллельный оператор, loop - итерация, opt - опциональная область.

Графически операторные конструкции изображаются в виде прямоугольника с пунктирными линиями в качестве разделителей. Ключевое слово оператора располагается в левом верхнем углу.

Альтернативная композиция (рис. 10.4) позволяет задавать альтернативное выполнение секций MSC-диаграммы. Только одна альтернатива может быть реализована.

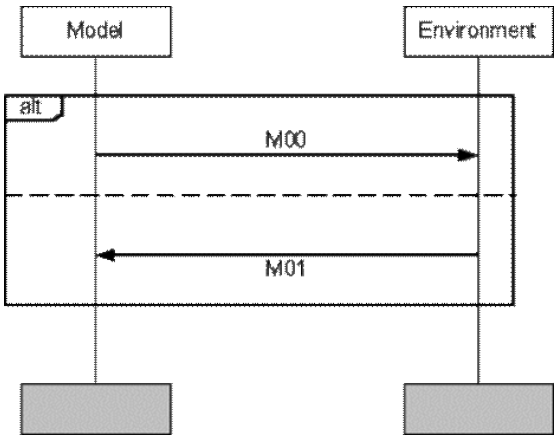


Рис. 10.4. Альтернатива

Операция par имеет структуру, аналогичную конструкции alt (рис. 10.3), и определяет параллельное выполнение секций. Это означает, что все события внутри параллельных секций будут выполнены. Единственным ограничением является то, что порядок событий в каждой секции будет сохранен.

Конструкция loop (рис. 10.5) имеет несколько форм. Наиболее общая форма - loop <n, m>, где n и m - натуральные числа. Это означает, что конструкция может быть выполнена от n до mраз. Вместо натурального числа может использоваться ключевое слово inf, обозначающее бесконечность.

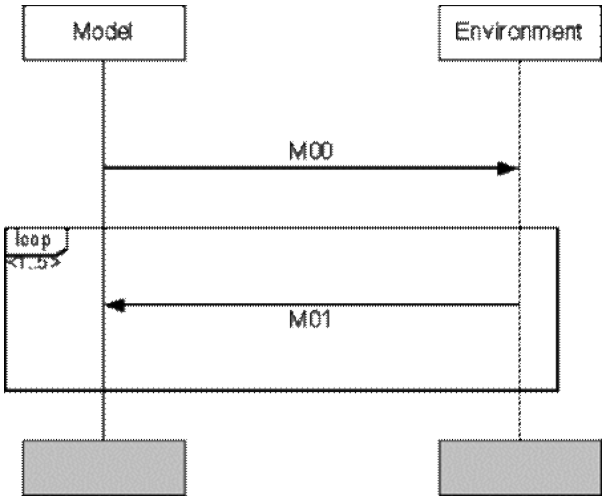


Рис. 10.5. Цикл

Оператор opt имеет структуру, аналогичную loop, но без операндов, и обозначает то же, что и оператор alt с пустой MSC в качестве второго операнда.

Одним из важных понятий в MSC является условие или состояние. Состояние - это особое событие на трассе объекта. В отличие от прочих событий, одно и то же состояние может разделяться одним, двумя и более объектами. По числу объектов на диаграмме, разделяющих некоторое состояние, различают глобальные состояния (общее для всех объектов), разделяемые состояния (разделяемые несколькими, но не всеми объектами) и локальные состояния (разделяемые единственным объектом). Если два объекта разделяют одно и то же состояние, то сопряженные события приема

и посылки сообщений должны происходить либо оба до соответствующего состояния, либо оба после соответствующего состояния.

Благодаря своему главному преимуществу - ясному графическому представлению, которое дает интуитивное понимание поведения описываемой системы, MSC-диаграммы широко применяются для различных целей:

- для определения требований;
- как спецификация интерфейсов;
- как спецификация взаимодействия процессов;
- как базис для генерации тестов;
- для документации;
- для объектно-ориентированного анализа и разработки.

Хотя изначально MSC-диаграммы предназначались для описания телекоммуникационных систем, сейчас они с успехом применяются и в других областях.

Обработка MSC-диаграмм

В случае достаточно сложных систем могут быть обнаружены ошибки, возникшие как при создании диаграммы, так и при проектировании тестируемой системы. Исправление тех же ошибок, если они будут обнаружены на более позднем этапе жизненного цикла продукта, потребует гораздо больших затрат.

Проверка MSC-диаграммы на полноту

Большинство систем или их частей можно в том или ином виде представить с помощью механизма state-машин. Тогда каждому состоянию системы на диаграмме будет соответствовать условие - конструкция condition. Для каждого события диаграммы (под событием будем понимать сообщение или действие - action) можно составить множество всех состояний, которые могут непосредственно предшествовать ему (предусловия). В этом случае проверка на полноту MSC-диаграммы будет заключаться в проверке того, все ли возможные случаи предусловий для каждого события представлены в ней. Если это не так, то, возможно, диаграмма не описывает полностью все возможные сценарии работы системы, и множество тестов, сгенерированных по этой диаграмме, будет неполным.

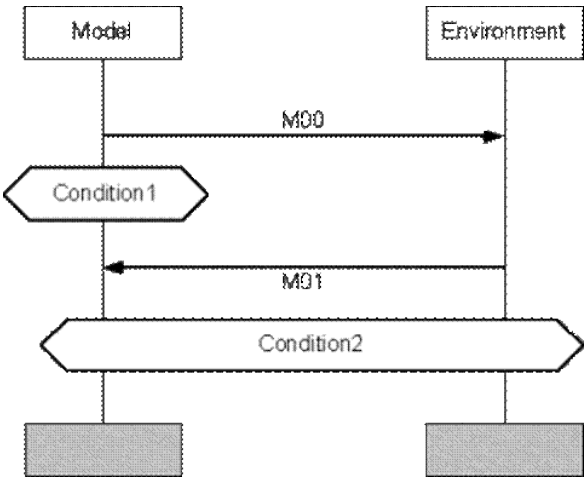


Рис. 10.6. Состояние (условие)

Основной недостаток стандартной MSC - невозможность описать отношения данных в параметрах сообщения. Эта проблема решается с использованием некоторого расширения стандартного MSC - mMSC (macro MSC). Основные из этих расширений включают в себя:

Макроподстановки (рис. 10.7). Они позволяют создавать множество MSC-диаграмм с одинаковой структурой и разными параметрами сообщения, циклами и т. п. Макроподстановки начинаются с символа # и могут быть константами или функциями. Функции, кроме названия, содержат параметры, заключенные в скобки.

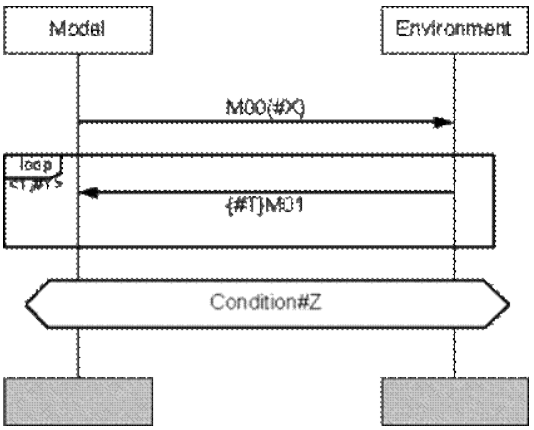


Рис. 10.7. Макроподстановки

Типы макроподстановок описываются в определенном файле, и на основе этой информации вместо них подставляются конкретные значения.

Временные ограничения служат для указания времени посылки/приема сообщения, его длительности и типа (синхронное с явно заданным моментом выполнения, асинхронное). Время может задаваться относительно начала работы (абсолютное), относительно предыдущего сообщения (относительное) или относительно метки. На рис. 10.8 сообщение M01 должно отправиться через 5 единиц времени, в течение 5 единиц после получения сообщения M00 (указано с помощью метки).

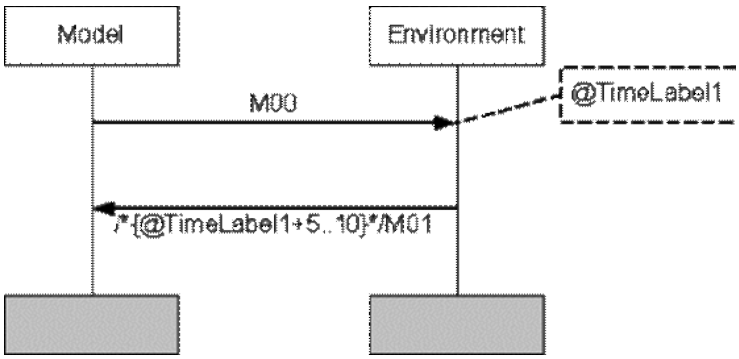


Рис. 10.8. Использование времени

Благодаря своему главному преимуществу - ясному графическому представлению, которое дает интуитивное понимание поведения описываемой системы, MSC-диаграммы широко применяются для различных целей:

- для определения требований;
- как спецификация интерфейсов;
- как спецификация взаимодействия процессов;
- как базис для генерации тестов;
- для документации;
- для объектно-ориентированного анализа и разработки.

Хотя изначально MSC-диаграммы предназначались для описания телекоммуникационных систем, сейчас они с успехом применяются и в других областях.

Обработка MSC-диаграмм

В случае достаточно сложных систем могут быть обнаружены ошибки, возникшие как при создании диаграммы, так и при проектировании тестируемой системы. Исправление тех же ошибок, если они будут обнаружены на более позднем этапе жизненного цикла продукта, потребует гораздо больших затрат.

Проверка MSC-диаграммы на полноту

Большинство систем или их частей можно в том или ином виде представить с помощью механизма state-машин. Тогда каждому состоянию системы на диаграмме будет соответствовать условие - конструкция condition. Для каждого события диаграммы (под событием будем понимать сообщение или действие - action) можно составить множество всех состояний, которые могут непосредственно предшествовать ему (предусловия). В этом случае проверка на полноту MSC-диаграммы будет заключаться в проверке того, все ли возможные случаи предусловий для каждого события представлены в ней. Если это не так, то, возможно, диаграмма не описывает полностью все возможные сценарии работы системы, и множество тестов, сгенерированных по этой диаграмме, будет неполным.

Лабораторная работа №22
Автоматическая генерация тестов на основе формального описания

Тесты составляются на основе спецификации требований. При формулировании требований на естественном языке существует проблема их различных толкований. Одним из способов избежать этого является применение формальных языков для описания структуры и поведения системы (UML, SDL, MSC). Кроме того, описание требований на формальном языке является формальным описанием тестовых случаев, на основе которого можно генерировать тестовый код. В практикуме для создания тестов будет использоваться язык диаграмм взаимодействия (Message Sequence Charts, MSC - п.11). В этом случае под тестом мы будем понимать его представление в виде MSC- диаграммы.

В Практикуме для реализации тестирования используется учебная система *автоматизации тестирования TAT - Test Automation Training*. На вход система принимает формальное описание тестов в виде MSC диаграмм (в текстовом формате MSC PR). На основе этих MSC диаграмм и конфигурационного файла (в формате XML), который описывает интерфейс тестируемой системы, генерируется тест на C#. (Интерфейс тестируемого приложения (Application Under Test - AUT) содержит сигналы, сообщения, транзакции, которые система может посылать тестовому окружению или может принимать от тестового окружения). Для запуска системы с этим тестом необходимо написать *Wrapper*, который транслирует сигналы от теста к системе и наоборот.

Таким образом, методика тестирования системы с помощью *TAT* выглядит следующим образом:

- написать *Wrapper* для тестируемой системы;
- создать файл конфигурации;
- создать формальное описание тестов в виде MSC-диаграмм;
- нарисовать MSC-диаграммы в MS Visio;
- сгенерировать с помощью макроса тестовый файл в формате *MPR*;
- настроить в ConfigTAT проект теста (указать пути) или набора тестов;
- запустить тест или набор тестов;
- проанализировать получаемые *log-файлы*.

В данном случае *wrapper* и файл конфигурации (первые два пункта методики) уже созданы, поэтому вам необходимо будет выполнить только п. 3-6.

В рассматриваемом подходе не только запуск тестов и проверка результатов прогона **тестового случая** будут осуществляться автоматически, но и сам тестовый код будет генерироваться автоматически на основе MSC-диаграммы (рис. 7.1).

При разработке тестов был использован следующий подход:

Когда реализуется определенное событие, модель посылает сигнал запроса состояния к тестовому окружению.

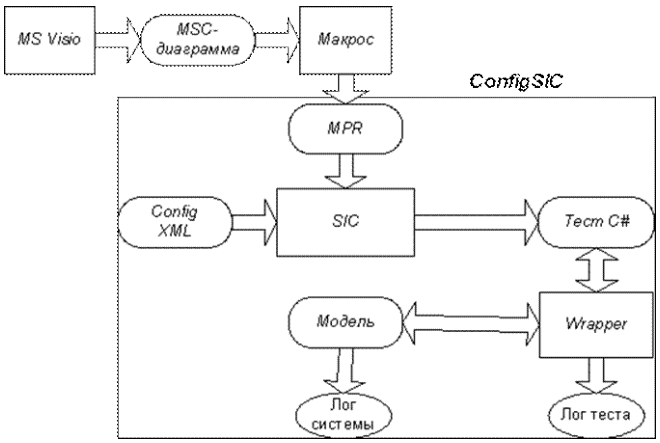


Рис. 7.1. Система и ее окружение (автоматическая генерация)

Состояние окружения задается в тесте (**входные данные**) в виде параметров сигналов. Тест возвращает состояние окружения (StoreStat, AxlePar, RollerPar, StoreMessage, CommandStatus), посылая модели сигнал с параметрами в соответствии с запросом.

Получаемая информация сохраняется в журнале теста.

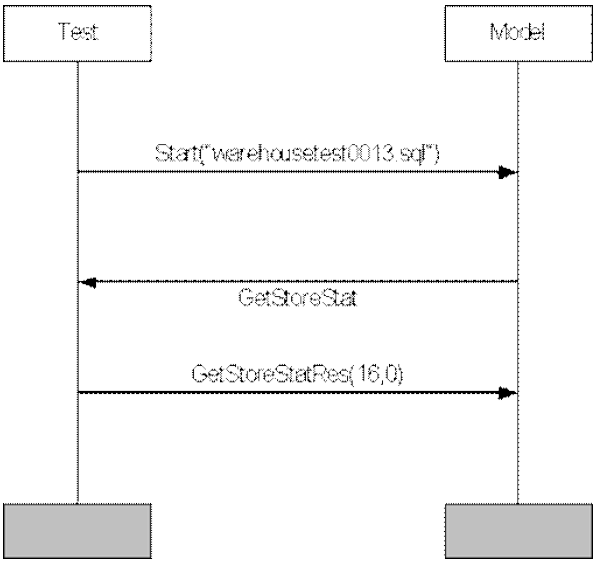


Рис. 7.2. Взаимодействие теста и модели (MSC-диаграмма)

Реализация выбранного подхода для ручного тестирования приводится в Class1.cs (..\SystemTesting\ManualTests\Tests\Class1.cs).

Все классы входят в пространство имен Tests. Это пространство имен содержит следующие классы:

- Class1 - главный класс приложения. Содержит статический метод Main, вызываемый при запуске;

- Test - абстрактный (abstract) класс, реализующий общую для всех тестов функциональность. Содержит следующие методы:

- public Test() - конструктор. Создает серверный сокет и запускает сервер;

- protected void wait(string st) - ожидает получения от dll вызова, начинающегося со строки st ;

- protected void finish() - обрабатывает последний запрос от dll и закрывает серверный сокет;

- virtual public void start() - запускает тест. В каждом конкретном тесте переопределяется.

Кроме того, класс Test содержит пять protected полей типа string:

- StoreStat - статус склада;
- AxlePar - терминал оси;
- RollerPar - терминал подшипника;
- CommandStatus - возвращаемое значение функции SendStoreCom ;
- StoreMessage - сообщение от склада.

Как создать свой тест?

Для создания нового теста необходимо выполнить следующие действия:

- создать новый класс, являющийся потомком Test ;
- переопределить в нем метод start(), чтобы реализовать функциональность теста:
- задать состояние окружения (StoreStat, AxlePar, RollerPar, StoreMessage, CommandStatus);
- ждать, когда произойдет определенное событие (вызов wait, в котором надо задать строку для выхода из состояния ожидания);
- задать новое состояние окружения и т.д.
- тест должен завершаться вызовом finish().

В общем виде тест выглядит так:

override public void start(){ <задание переменных>wait(<строка>);<задание переменных>wait(<строка>);....finish();}

Лабораторная работа № 23
«Автономная отладка ИС»

Для отладки проекта ИС снабжена следующими механизмами:

- автономная отладка шаблонов;
- запуск выделенного узла (слоя **Технология**) под управлением одного из отладочных мониторов (профайлеров) из интегрированной среды по команде **Отладка** с протоколированием работы в текстовый файл;
- использование функции **шпион** – эта функция обеспечивает получение в режиме редактирования реальных данных с работающих узлов проекта. Для ее использования нужно выполнить команду **Шпион**

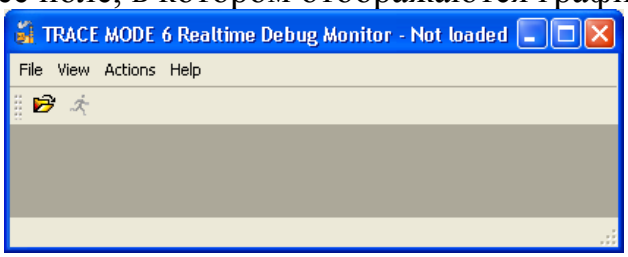
Профайлеры

Для отладки узла (слоя **Технология**) его можно запустить (в том числе из ИС) под управлением одного из следующих отладочных мониторов:

- профайлера с поддержкой графических экранов (**rtc.exe**);
- профайлера без поддержки графических экранов (**rtmg32.exe**).
- При конфигурировании ИС можно указать профайлер, который должен запускаться из ИС.
- Для профайлера без поддержки графики можно задать режим отображения каналов распределенными по внутренним стандартным группам (объектам) TRACE MODE.
- Профайлеры записывают протокол своей работы в файл **<имя файла prj>_<порядковый номер узла>.txt**, который сохраняется в папке узла. Степень детализации отладочной информации, выводимой в файл, может быть задана.

Профайлер с поддержкой графических экранов

Графическая оболочка этого профайлера содержит меню, панель инструментов и рабочее поле, в котором отображаются графические экраны:

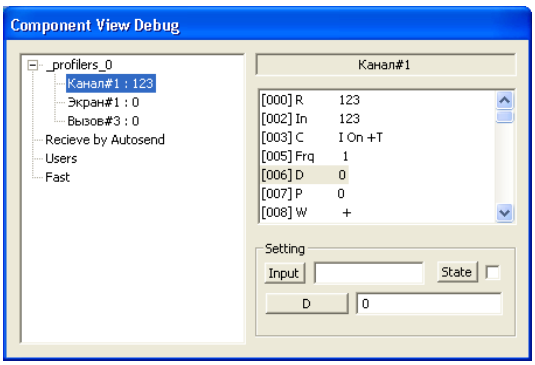


Меню **Файл** и панель инструментов содержат команды открытия (, CTRL+O), перезагрузки и запуска (, CTRL+R) узла, а также команду выхода из программы.

Меню **Операции** содержит команды авторизации/окончания сеанса, а также команду посылки сообщения в отчет тревог.

Меню **Вид** содержит следующие команды:

- **Полный экран** (CTRL+F) – переключение вида отображения графический экранов (в окне/полноэкранный);
- **Компоненты** (CTRL+0) – по этой команде на экране появляется диалог, в левой части которого отображаются каналы узла, а правая содержит инструменты задания атрибутов канала, выделенного в левой части:



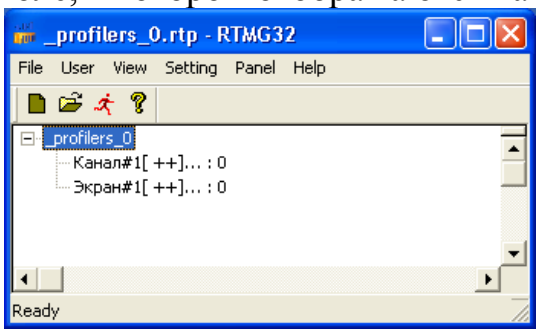
Для задания входного значения канала (атрибута 0, **In**) нужно ввести требуемое значение в поле справа от кнопки **Input** и нажать ЛК на этой кнопке.

Для задания значения произвольного атрибута нужно выделить атрибут в списке, ввести требуемое значение в поле справа от кнопки, на которую выводится короткое имя выбранного атрибута, и нажать ЛК на этой кнопке.

Для выключения канала нужно установить флаг в поле справа от кнопки **State** и нажать ЛК на этой кнопке.

Профайлер без поддержки графических экранов

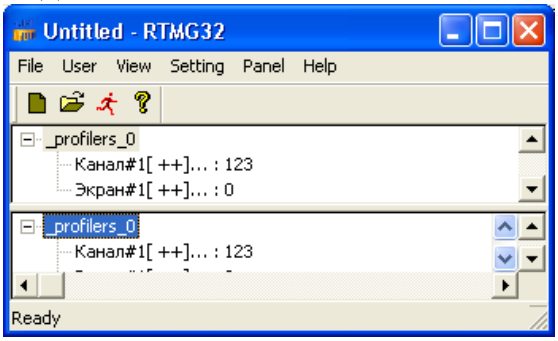
Графическая оболочка этого профайлера содержит меню, панель инструментов и рабочее поле, в котором отображаются каналы узла:



Меню **Файл** и панель инструментов содержат команды открытия (, CTRL+O), перезагрузки (, CTRL+R) и запуска ( ) узла, а также команду выхода из программы.

Меню **Пользователь** содержит команды авторизации/окончания сеанса, а также команду **Пользователи**.

Меню **Вид** содержит флаги управления видимостью строки статуса, панели инструментов и разделения окна:



5. Программирование алгоритмов в TRACE MODE 6

Для программирования алгоритмов функционирования разрабатываемого проекта АСУ в TRACE MODE 6 включены языки **Техно**

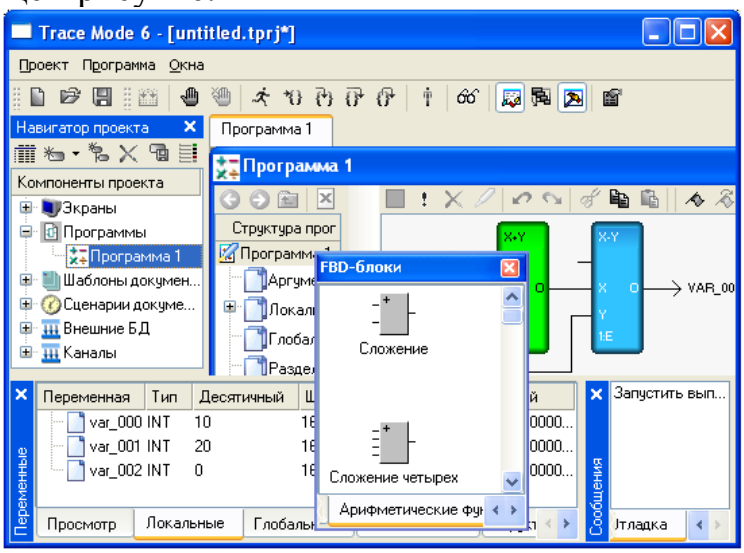
ST, Техно SFC, **Техно FBD, Техно LD** и **Техно IL**. Данные языки являются модификациями языков **ST** (Structured Text), **SFC** (Sequential Function Chart), **FBD** (Function Block Diagram), **LD** (Ladder Diagram) и **IL** (Instruction List) стандарта IEC61131-3.

Программы и некоторые их компоненты (функции, шаги и переходы SFC и т.п.) могут быть разработаны на любом из встроенных языков в соответствующем редакторе, при этом языки для программы и ее компонентов выбираются независимо.

Для создания и редактирования свойств аргументов, переменных, функций и структурных типов программы, а также для использования в программе функций из внешних библиотек в интегрированную среду разработки проекта встроены специальные табличные редакторы.

TRACE MODE 6 имеет также средства для отладки программ.

Примерный вид интегрированной среды при редактировании программ показан на следующем рисунке:



Основным языком программирования TRACE MODE 6 является **Техно ST.** Программы, разработанные на языках **Техно LD**, **Техно SFC** и **Техно FBD**, перед компиляцией транслируются в **Техно ST**. **IL**-программы перед компиляцией частично транслируются в **ST**, частично – в ассемблер. Отсюда следует, например, что ключевые слова **Техно ST** являются таковыми и для всех других языков.

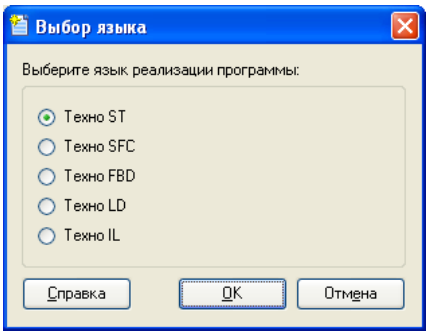
Подключение программы к проекту

Для подключения программы к проекту ее нужно вначале скомпилировать, а затем сохранить проект. Чтобы скомпилировать программу, нужно выполнить команду **Компилировать** из меню **Программа**, или нажать клавишу **F7** или нажать ЛК на иконке  панели инструментов отладчика. Перед сохранением проекта нужно убедиться, что компиляция прошла успешно (в окне сообщений компилятора в этом случае выводится соответствующее сообщение).

Отладка программы также возможна только после ее успешной компиляции.

Выбор языка программирования

Язык программирования может быть независимо задан для основной программы, функции-блока, функции и шага SFC. Язык выбирается в следующем диалоге:



Этот диалог автоматически появляется на экране при нажатии ЛК на имени вновь созданной программы или ее компонента (для которого язык может быть задан независимо) в окне структуры программы. После выбора языка программа (компонент) открывается в соответствующем редакторе.

Изменить язык можно только после удаления тела программы (компонента). Для этого нужно нажать ЛК на иконке  панели инструментов в окне структуры программы, после чего диалог выбора языка автоматически появляется на экране.

Создание элементов программ с помощью табличных редакторов

Табличные редакторы используются для создания следующих компонентов и элементов программ:

- аргументы;
- локальные переменные;
- глобальные переменные;
- функции-блоки (подпрограммы) и функции;
- структурные типы.
- Кроме того, с помощью табличных редакторов конфигурируются обращения к функциям из внешних библиотек.
- Перечисленные компоненты и элементы, наряду с листингами **ST** и **IL** и диаграммами **LD**, **SFC** и **FBD**, образуют ветви дерева в окне структуры программы.
- Для входа в соответствующий табличный редактор нужно в окне структуры программы нажать ЛК на любом из перечисленных выше элементов.

Особенности редактирования

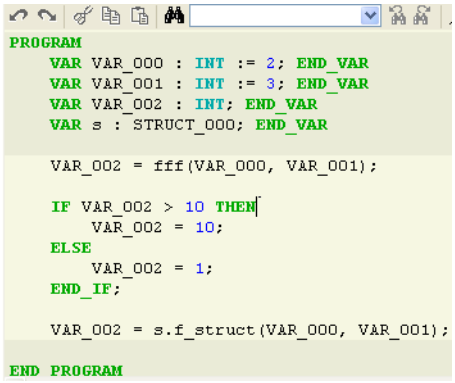
Для создания/удаления строк и поиска в табличных редакторах используется типовая панель инструментов.

Для перехода к редактированию отдельной ячейки таблицы нужно дважды нажать ЛК на этой ячейке. Редактирование ячейки производится либо путем непосредственного ввода с клавиатуры, либо путем выбора нужного значения из списка.

При задании числа в качестве разделителя целой и дробной части используется точка.

Если в ячейку столбца **[] Массив** ввести число, равное количеству элементов массива, то в этой ячейке отобразится диапазон индексов элементов (начиная с 0). Например, для двумерного массива при вводе **9, 8** отобразится **0 .. 8, 0 .. 7**.

Некоторые элементы (например, переменные), заданные в табличных редакторах, автоматически добавляются в листинги текстовых программ в виде соответствующих конструкций языка. Эти конструкции выделяются серым цветом; они недоступны для непосредственного редактирования с помощью клавиатуры:



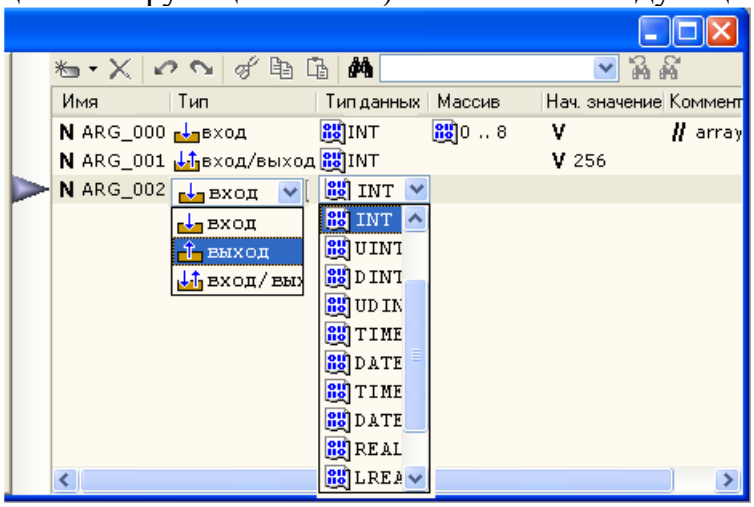
```
PROGRAM
    VAR VAR_000 : INT := 2; END_VAR
    VAR VAR_001 : INT := 3; END_VAR
    VAR VAR_002 : INT; END_VAR
    VAR s : STRUCT_000; END_VAR

    VAR_002 = fff(VAR_000, VAR_001);

    IF VAR_002 > 10 THEN
        VAR_002 = 10;
    ELSE
        VAR_002 = 1;
    END_IF;

    VAR_002 = s.f_struct(VAR_000, VAR_001);

END PROGRAM
```

Доступные типы данных (столбец **Тип данных**) для программ на всех языках одинаковы.

Начальное значение (столбец **Начальное значение**) может быть задано в любой из форм, определенных для **Техно ST**.

Табличный редактор аргументов программного компонента

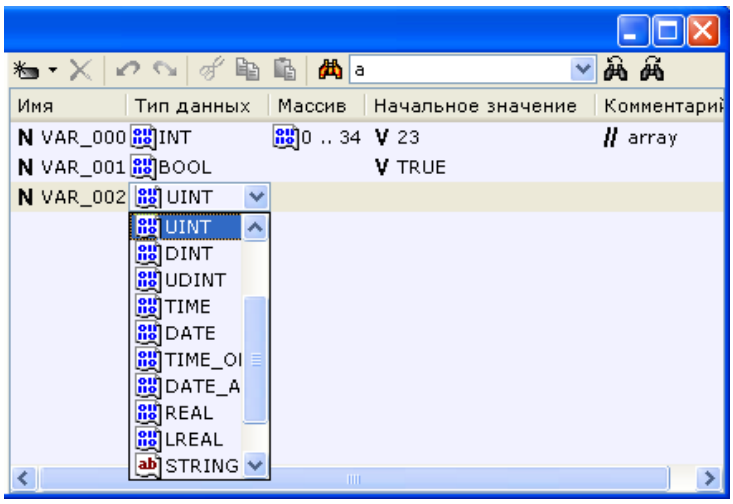
Вид табличного редактора аргументов программного компонента (функции или функции-блока) показан на следующем рисунке.



В этом редакторе задается имя аргумента, его тип (**вход**, **выход** или **вход/выход**), тип данных, начальное значение и комментарий. Если в поле **Массив** строки аргумента задать число, аргумент интерпретируется как массив.

Табличный редактор переменных

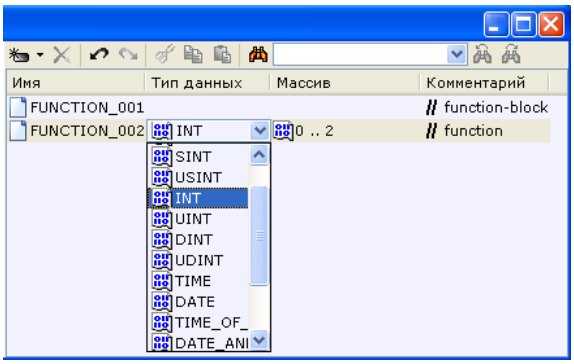
Вид табличного редактора переменных показан на следующем рисунке



В этом редакторе задается имя переменной, ее тип данных, начальное значение и комментарий. Если в поле **Массив** строки переменной задать число, переменная интерпретируется как массив.

Табличный редактор функций и функций-блоков

Вид табличного редактора функций и функций-блоков показан на следующем рисунке.



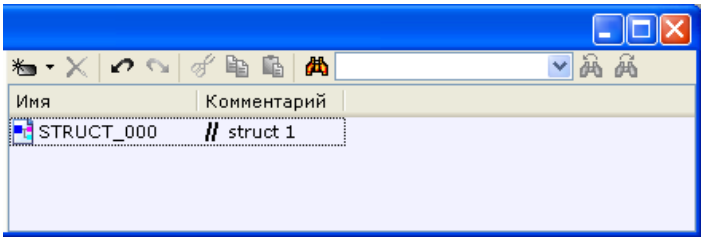
В этом редакторе задается имя функции (функции-блока) и комментарий.

Если указан тип возвращаемого значения, определяется функция, если тип возвращаемого значения не указан, определяется функция-блок.

Если в поле **Массив** строки функции задать число, функция возвращает массив. Для функции-блока поле **Массив** недоступно.

Табличный редактор структурных типов

В этом редакторе задается имя создаваемого структурного типа и комментарий.

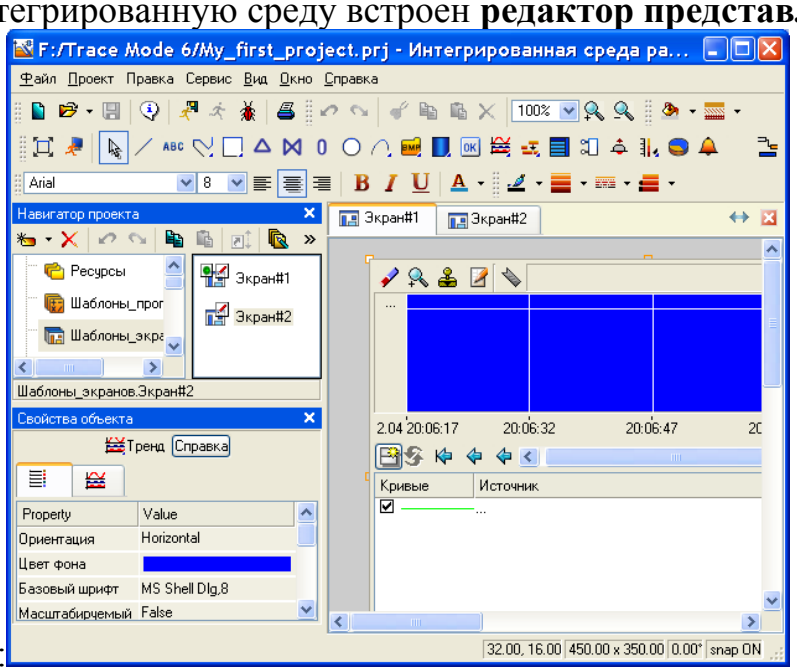


Более подробное описание алгоритмов программирования, а также описание языков программирования можно найти в справочной системе TRACE MODE в разделе **Программирование алгоритмов**.

6. Разработка графического интерфейса

Редактор представления данных

Графическое представление хода выполнения техпроцесса, а также управление техпроцессом с помощью графических средств являются одними из главных задач, решаемых ТРЕЙС МОУД 6. Для разработки интерфейса оператора в интегрированную среду встроен **редактор представления данных (РПД)**:



Интерфейс оператора разрабатывается в виде набора **графических экранов**, являющихся компонентами проекта.

С целью взаимодействия с другими компонентами проекта для графического экрана могут быть заданы аргументы.

Совокупность графических экранов узла образует его **графическую базу**. Совокупность графических баз всех узлов разрабатываемого проекта АСУТП образует **графическую часть** проекта.

Графический экран может содержать один или несколько графических **слоев**, каждый из которых, в свою очередь, может содержать один или несколько **подслоев**.

В графических слоях размещаются **графические элементы (ГЭ)**. Графические элементы имеют наборы настраиваемых **атрибутов**, **динамических свойств** и **функций управления**. Эти параметры определяют вид графических элементов и выполняемые ими функции отображения и управления при работе в реальном времени. Редактор представления данных содержит большое количество встроенных графических элементов, позволяющих изобразить практически любой техпроцесс, вывести на дисплей всю необходимую информацию о ходе его выполнения, а также управлять техпроцессом.

Режимы работы РПД

Редактор представления данных может находиться в одном из следующих режимов:

- режим **размещения** предназначен для заполнения графических слоев экранов

графическими элементами. Для перехода в этот режим нужно нажать одну из кнопок выбора ГЭ на панели инструментов **Графические элементы**;

- режим **редактирования** предназначен для внесения изменений в созданные ранее графические экраны (например, для удаления/добавления графических элементов или изменения их свойств). Для перехода в этот режим надо нажать кнопку  главной панели инструментов;
- режим **эмуляции** используется для проверки работы графических элементов в реальном времени. Для перехода в режим эмуляции надо нажать кнопку  панели инструментов **Графические элементы**; для выхода из режима надо нажать кнопку  повторно.
- Кроме того, в РПД предусмотрено два режима отображения графических экранов – обычный (в окне) и полноэкранный. Для переключения режима отображения используется кнопка  панели инструментов **Графические элементы**.
- В режимах размещения и редактирования текущие координаты курсора отображаются в строке статуса (внизу справа). Там же отображается состояние флага **Располагать по сетке**.

Главное меню и панели инструментов РПД

Панель инструментов 'Графические элементы'

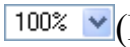
С помощью инструментов этой панели выбираются графические элементы для размещения их в графических слоях экранов. При выборе ГЭ редактор переходит в режим размещения.

С помощью кнопки  данной панели можно перейти в режим редактирования, с помощью кнопки  – в режим эмуляции.

Кнопка  предназначена для переключения режима отображения графических экранов (обычный/полноэкранный).

Меню и панель инструментов 'Правка'

Меню и панель инструментов **Правка** содержат ряд типовых инструментов для редактирования графических экранов. Данные инструменты доступны также из контекстного меню ГЭ.

В списке 

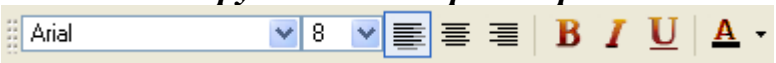
(**Масштаб**), а так же при помощи кнопок  и  панели инструментов **Правка** выбирается масштаб отображения.

Меню 'Сервис' и панель инструментов 'Топология экрана'

Данные панель инструментов и меню содержат команды для позиционирования и тиражирования выделенного графического элемента

Меню **Сервис** содержит дополнительно команду **Параметры экрана**.

Панель инструментов 'Параметры текста'



В режиме редактирования с помощью типовых инструментов данной панели задаются параметры текста в выделенном графическом элементе (выделенной группе ГЭ). Данные команды применимы только к такому тексту, который может быть введен/отредактирован с помощью клавиатуры.

Вид ГЭ при его размещении в графическом слое зависит от параметров, установленных с помощью инструментов этой панели.

Панель инструментов 'Параметры линии'

В режиме редактирования с помощью инструментов этой панели задаются параметры линии (линии контура) выделенного графического элемента (выделенной группы ГЭ):

 – выбор **цвета линии**. По этой команде на экран выводится стандартный диалог выбора цвета;

 – выбор **толщины линии**.

 – выбор **стиля линии**. По этой команде открывается список стилей, содержащий в том числе опцию **Без линии**

 – выбор **края линии** (**плоский**, **квадратный**, **круглый**).

Вид ГЭ при его размещении в графическом слое зависит от параметров, установленных с помощью инструментов этой панели.

Панель инструментов 'Параметры заливки'

В режиме редактирования с помощью инструментов этой панели задаются параметры заливки выделенного графического элемента (выделенной группы ГЭ

 – выбор **цвета заливки**. По этой команде на экран выводится стандартный диалог выбора цвета;

 – выбор **стиля заливки**.

Вид ГЭ при его размещении в графическом слое зависит от параметров, установленных с помощью инструментов этой панели.

Панель инструментов 'Ресурсные

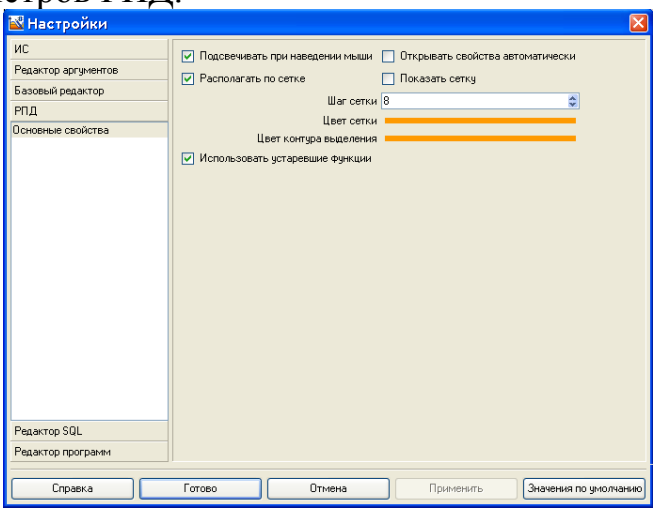
Инструменты данной панели предназначены для операций с библиотеками строк, рисунков и других ресурсов, которые могут быть использованы при разработке графических экранов.

Меню 'Вид'

Команды этого меню управляют видимостью табличного редактора аргументов экрана, окна **Слои** и таблицы графических элементов, а также панелей инструментов **Топология экрана** и **Параметры текста**.

Задание параметров РПД

Окно **Настройки**, вызываемое из меню **Файл**, в разделе **РПД** содержит диалог задания параметров РПД.



Этот диалог содержит следующие инструменты:

Подсвечивать при наведении мыши – если этот флаг установлен, при наведении курсора мыши на ГЭ его вершины (узловые точки) выделяются красным цветом. Не следует путать эту функцию с функцией выделения ГЭ (.

Открывать свойства автоматически – от этого флага зависит режим РПД после размещения графического элемента на экране;

Располагать по сетке – если этот флаг установлен, при размещении, перемещении и масштабировании вершины прямоугольника, ограничивающего ГЭ, располагаются в узлах сетки. При размещении в узлах сетки располагаются также узловые точки ГЭ.

Показать сетку – если этот флаг установлен, сетка отображается на графических экранах;

Шаг сетки – задание шага сетки в пикселях (1-100);

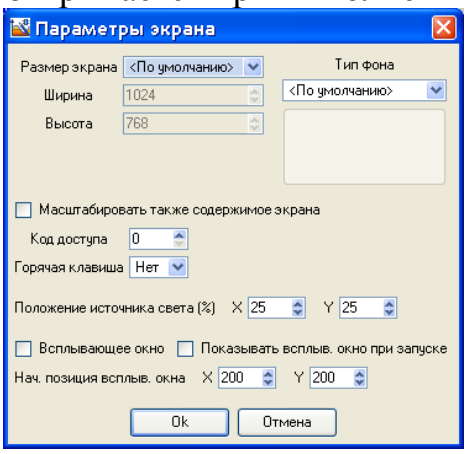
Цвет сетки – выбор цвета сетки;

Цвет контура выделения – выбор цвета прямоугольника, ограничивающего ГЭ при выделении.

Использовать устаревшие функции – при установке этого флага доступны некоторые опции предыдущих версий ТРЕЙС МОУД.

Задание параметров графического экрана

Параметры редактируемого графического экрана задаются в диалоге, который открывается при выполнении команды **Параметры экрана** меню **Сервис**:



Этот диалог содержит следующие инструменты:

Размер экрана – задание размера экрана в пикселях. Размер можно выбрать из нескольких стандартных или задать свой с помощью опции **Произвольный** и полей **Ширина** и **Высота**.

- **Тип фона** – выбор типа фона. Содержит следующие варианты:
- **<По умолчанию>** – фон экрана по умолчанию;
- **Цвет** – при выборе этой опции для экрана можно задать цвет фона – для этого нужно нажать кнопку под списком и выбрать цвет в стандартном диалоге;
- **Изображение** – при выборе этой опции в качестве фона экрана можно использовать рисунок;
- **Масштабировать также содержимое экрана** – флаг, при установке которого размещенные на экране ГЭ масштабируются пропорционально изменению размеров экрана;
- **Код доступа** – код доступа к экрану (0-255). Права на доступ к экранам задаются для пользователя в виде маски в разделе **Доступ** / **Экраны** канала **Пользователь**. При корреляции маски с кодом доступа (результат побитового логического умножения отличен от нуля) доступ к экрану разрешен, в противном случае – запрещен;
- **Горячая клавиша** – меню выбора функциональной горячей клавиши (**F2** – **F12**). При запуске проекта в реальном времени нажатие заданной клавиши будет сопровождаться переходом на данный экран;
- **Положение источника света (%)** – положение источника света относительно экрана (угол с осями X и Y в процентах). Значение (50, 50) соответствует расположению источника света на нормали к экрану;
- **Всплывающее окно** – показывать экран в списке всплывающих экранов МРВ;
- **Показывать всплыв. окно при запуске** – показывать экран при загрузке узла в МРВ;
- **Нач. позиция всплыв. окна , X** и **Y** – положение всплывающего экрана по осям X и Y при загрузке узла в МРВ. Данная опция не влияет на отображение обычного экрана.

Часть параметров для создаваемых графических экранов можно задать в редакторе группы шаблонов экранов.

Задание аргументов графического экрана

Для графического экрана могут быть заданы аргументы с целью взаимодействия с другими компонентами проекта. Чтобы открыть табличный редактор аргументов экрана, нужно выбрать опцию **Аргументы экрана** в меню **Вид**.

Операции с графическими элементами

Размещение ГЭ

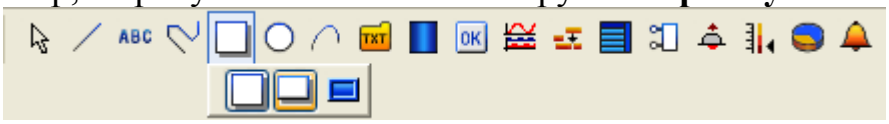
Встроенные графические элементы разбиты на группы. Каждой группе соответствует кнопка на панели инструментов **Графические элементы**. На кнопку выводится иконка одного из элементов данной группы.

Чтобы выбрать ГЭ для размещения, нужно выполнить следующие действия:

нажать ЛК на кнопке панели инструментов **Графические элементы**. При этом выбирается тот элемент, чья иконка выведена на кнопку (элемент, заданный по умолчанию для соответствующей группы, или элемент, выбранный ранее);

- дважды нажать ЛК на кнопке и затем нажать ЛК на иконке требуемого ГЭ в появившемся меню (меню не открывается, если группа содержит только один графический элемент).

После выбора элемента его иконка выводится на кнопку группы. Например, на рисунке показано меню группы **Прямоугольники**:



После выбора элемента его иконка выводится на кнопку группы:

При выборе графического элемента редактор представления данных переходит в режим размещения, при этом курсор на графическом экране приобретает вид +.

Далее в окне **Слои** необходимо нажатием ЛК указать слой, в котором должен быть размещен выбранный графический элемент.

Далее продолжить процедуру размещения ГЭ можно двумя способами:

- перетащить ГЭ с панели инструментов на экран (метод **drag-and-drop**); после размещения ГЭ имеет размеры, заданные по умолчанию, РПД переходит в режим редактирования, окно свойств ГЭ открывается автоматически;
- переместить курсор в нужную точку экрана и нажатием ЛК установить **точку привязки** ГЭ. Далее действия по размещению ГЭ могут отличаться, однако для большинства графических элементов они стандартны – перемещение мыши после установки точки привязки выводит на экран образ ГЭ, при этом отрезок от точки привязки до текущего положения курсора является диагональю прямоугольника, ограничивающего ГЭ. (Если при перемещении

мыши удерживать нажатой клавишу **CTRL**, ряд ГЭ окажется вписанным в квадрат). Повторное нажатие ЛК приводит к размещению графического элемента в выбранном графическом слое.

Для графических элементов групп **Ломаные** и **Кривые** каждое нажатие ЛК после установки точки привязки задает **узловую точку** (промежуточную вершину). Для установки последней вершины и выхода из режима размещения этих ГЭ нужно нажать ПК. Положение узловых точек, заданное при размещении, можно в дальнейшем изменить.

Режим РПД после размещения ГЭ данным способом зависит от флага **Открывать свойства автоматически**:

-если флаг установлен, то после размещения ГЭ автоматически открывается окно его свойств, а РПД переходит в режим редактирования;

- если флаг не установлен, то после размещения ГЭ РПД остается в режиме размещения. Этот способ удобен для многократного размещения на экране одного и того же графического элемента.

Перемещение и масштабирование ГЭ

Для перемещения или изменения размеров выделенного ГЭ (группы ГЭ) нужно выбрать в контекстном меню графического элемента режим **Перемещать/масштабировать** и далее использовать стандартные операции редактирования.

Удаление ГЭ

Для удаления выделенного ГЭ (группы ГЭ) нужно нажать клавишу **Del** на клавиатуре или выполнить команду **Удалить** с помощью меню или панели инструментов **Правка** или с помощью контекстного меню ГЭ.

В полноэкранном режиме редактирования для удаления выделенного ГЭ (группы ГЭ) с помощью клавиатуры используется комбинация клавиш **Ctrl+Del**.

Копирование и вставка ГЭ

Для копирования выделенного ГЭ (группы ГЭ) в буфер обмена нужно выполнить команду **Копировать**.

Для вставки содержимого буфера обмена в слой нужно выполнить команду **Вставить**.

Скопировать в буфер обмена можно в том числе группу графических элементов, лежащих в разных слоях экрана, однако при вставке такой группы все ее ГЭ будут размещены в одном и том же слое.

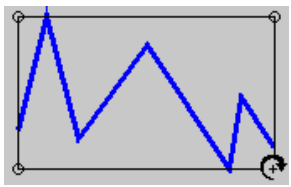
Выполнить команды **Копировать** и **Вставить** можно с помощью меню или панели инструментов **Правка** или с помощью контекстного меню ГЭ.

Поворот ГЭ

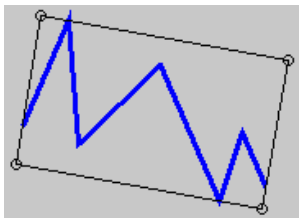
Существует 2 способа поворота ГЭ на экране в режиме редактирования.

Режим ’Повернуть’

Для перехода в этот режим нужно выбрать в контекстном меню выделенного ГЭ (группы ГЭ) опцию **Повернуть**. Далее надо установить курсор в одну из вершин прямоугольника, ограничивающего ГЭ (группу ГЭ) (курсор при этом принимает вид ):



Затем нужно нажать ЛК и, удерживая кнопку нажатой, перемещением мыши задать нужный угол поворота ГЭ. Для выхода из режима надо отпустить ЛК:



При использовании данного метода ГЭ (группа ГЭ) вращается относительно центра ограничивающего прямоугольника (центром прямоугольника является точка пересечения его диагоналей).

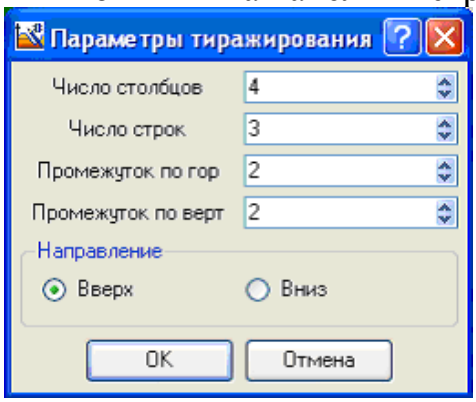
Поворот из таблицы графических элементов

Угол поворота ГЭ можно задать в поле **Угол (градусы)** диалога **Геометрия**. При использовании этого метода ГЭ поворачивается относительно точки привязки (при задании положительного значения – по часовой стрелке).

Тиражирование ГЭ

Тиражирование – это копирование выделенного графического элемента и его множественная вставка с табличным упорядочением. Копии вставляются вправо и вверх/вниз относительно выделенного ГЭ. Тиражирование выделенной группы ГЭ не поддерживается.

Операция тиражирования конфигурируется в диалоге, который открывается при выполнении команды **Тиражировать** из меню **Сервис** или нажатии кнопки  на панели инструментов **Топология экрана**:

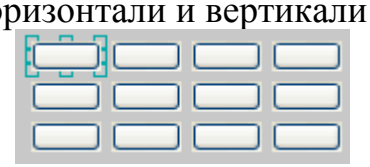


В этом диалоге задаются следующие параметры:

- **Число столбцов** – число элементов в строке (включая первоначально выделенный ГЭ);
- **Число строк** – число строк;

- **Промежуток по горизонтали** – промежуток между копиями по горизонтали в пикселях;
- **Промежуток по вертикали** – промежуток между копиями по вертикали в пикселях;
- **Вверх** – размножение по строкам вверх относительно выделенного ГЭ;
- **Вниз** – размножение по строкам вниз относительно выделенного ГЭ.

Ниже показан результат тиражирования кнопки (выделена на рисунке) при следующих параметрах: число столбцов – 4, число строк – 3, промежуток по горизонтали и вертикали – 5, направление тиражирования – вниз:



ЛАБОРАТОРНАЯ РАБОТА №24
Лабораторная работа «Комплексная отладка ИС»

Цель работы: получить начальное представление о возможностях отладчика.

1. **Метод**

ические указания

Одной из важнейших проблем, связанных с программированием, является проблема своевременного обнаружения и устранения ошибок, возникающих при создании программ. Человечество подходит к разрешению этой проблемы с разных сторон. Во-первых, создаются методы и средства автоматизации программирования, позволяющие уменьшить вероятность возникновения ошибки как таковой. К их числу относятся технологии структурного программирования и само объектно-ориентированное программирование. Во вторых, совершенствуются сами программные средства и языки программирования, создаются специализированные программы, которые позволяют относительно легко обнаруживать ошибки. Наконец, в третьих, ведутся статистические исследования и выявляются типовые ошибки, которые делают программисты. На основании полученной статистики выдаются рекомендации разработчикам нового программного обеспечения. Настоящая лабораторная работа посвящена изучению методов обнаружения ошибок, реализованному в VBA.

Обычно выделяют три основных вида ошибок, которые приводят к неправильному выполнению программы или делают ее выполнение просто невозможным. Первый вид ошибок – это ошибки, возникающие на этапе компиляции. Основной смысл определяемых на этапе компиляции ошибок – это некорректная запись операторов программы с точки зрения правил языка программирования. Как следствие, компилятор не может создать код и требует внести изменения в программу. Компилятор VBA высвечивает строку программы, которая содержит ошибку, красным цветом и выдает дополнительное диагностическое сообщение.

Ошибки этапа компиляции устраняются программистом с относительно небольшими затратами труда, поскольку их поиск автоматизирован, а для уточнения правил языка программист может легко воспользоваться справочной литературой или встроенной в компилятор VBA системой помощи. Ее вызов осуществляется при нажатии клавиши F1. С целью минимизации вероятности возникновения орфографической ошибки при записи класса или метода объекта может быть вызвано специальное контекстное меню вводом команды *Редактирование/Список свойств и методов*. Аналогичное меню может быть вызвано и для списка констант. Быстрый вызов меню можно осуществить и правой клавишей мыши при наборе текста. Наконец, распознанные операторы языка выделяются цветом, что позволяет уменьшить вероятность ошибки, связанной с неправильным именем переменной. В любом случае ошибки компиляции сопровождаются диагностическим сообщением, из которого, воспользовавшись при необходимости системой помощи, можно установить их причину.

Более сложный класс ошибок – это ошибки, возникающие на этапе выполнения программы. Эти ошибки в том или ином виде связаны с обрабатываемыми данными и, как следствие, не могут быть определены на этапе компиляции, поскольку конкретные значения данных в этот момент неизвестны. При возникновении подобных ошибок на экран выдается диагностическое сообщение с указанием кода ошибки и его кратким описанием. Составляя алгоритм, программист обязан предусмотреть возможность их появления и принять дополнительные меры по их локализации и, если это требуется, перехвату. Список некоторых ошибок этапа выполнения приведен в табл. 2.1.

Наиболее сложным видом ошибок при программировании являются алгоритмические ошибки. Причина таких ошибок двояка – с одной стороны они возникают из-за неправильного составления алгоритма, с другой из-за неправильного кодирования (записи операторов программы не в соответствии с составленным алгоритмом). К сожалению, единственным способом обнаружения алгоритмических

ошибок является тестирование. Под тестированием обычно понимают испытание программы при условии подачи на нее заведомо известных данных (теста) и проверки результатов ее работы (они должны быть определены совместно с подготовкой теста). Особенностью тестирования является то обстоятельство, что если тест обнаруживает факт существования алгоритмической ошибки (программа выполняется неверно), то ошибка существует и должна быть устранена. В тоже время, если тест не находит ошибки, то это обстоятельство не является доказательством того, что ошибка отсутствует. Как следствие, созданная программа должна быть подвергнута максимально возможному тестированию. Однако исчерпывающее тестирование программы, как правило, является невозможным из-за чрезвычайно большого числа возможных вариантов данных, в связи с чем приходится использовать методы программирования, уменьшающие вероятность возникновения ошибки, и рассчитывать на искусство программиста. Если в процессе тестирования была обнаружена ошибка, программист должен начать процесс определения конкретных операторов программы, вызвавших появление ошибки, обычно называемый отладкой. Для автоматизации процесса тестирования и отладки созданы специальные программы, которые получили название программ-отладчиков. Подобная программа есть и в составе редактора VBA.

Таблица 2.1.

Ошибки этапа выполнения

Код ошибки	Диагностическое сообщение
5	Приложение не запущено
6	Переполнение
7	Не хватает памяти
9	Выход индекса за границы диапазона
11	Деление на нуль
13	Несоответствие типов данных
18	Произошло прерывание, вызванное пользователем
52	Неправильное имя файла или идентификатора
53	Файл не найден
54	Неправильный режим работы с файлом
55	Файл уже открыт
56	Ошибка ввода-вывода
61	Переполнение диска
68	Устройство не доступно
71	Диск не готов
72	Повреждена поверхность диска
335	Невозможен доступ к системным ресурсам

Код ошибки	Диагностическое сообщение
368	Истек срок действия данного файла
482	Ошибка принтера

Рис. 4. Меню отладчика.

Режим отладки включается из главного меню при активном окне редактора при выполнении команды *Отладка/Шаг с заходом*. Выключение режима обеспечивает выполнение команды *Выполнить/Сброс*. Внешний вид меню показан на рис. 4. Обратите внимание на то, что основные команды отладчика вызываются комбинациями клавиш F8 и F9. Команда *Шаг с заходом* позволяет оператор за оператором выполнить тестируемую программу, включая вызываемые программой функции и процедуры. Команда *Шаг с обходом* исключает пошаговое выполнение вызываемых модулей. Команда *Шаг с выходом* завершает пошаговое выполнение вызванного модуля. Команда *Выполнить до текущей позиции* выполняет программу до оператора, на котором установлен курсор. Команда *Точка останова* задает и снимает точку останова в тексте программы, причем конкретный оператор предварительно выбирается курсором. Команда *Снять все точки останова* удаляет все установленные в программе точки останова. Выполнение команд *Задать следующую инструкцию* и *Показать следующую инструкцию* позволяет найти следующий выполняемый оператор в окне редактирования.

Текущие значения переменных можно наблюдать задавая их имена в окне контрольных значений (*Отладка/Добавить контрольные значения, Отладка/Контрольные значения, Вид/Окно локальных переменных*). При выполнении команды *Вид/Проверка* можно задать дополнительный оператор VBA, или изменить значение любой переменной оператором присваивания.

2. Порядок выполнения работы

1. Включите компьютер. Загрузите *Windows*. Загрузите *Excel* и выполните команду *Файл/Открыть*. При стандартной настройке на экране появится окно *Открытие документа*, а в окне *Папка* появится название основной рабочей папки *Мои документы*. Если по каким-то причинам установилась другая папка, нажмите кнопку меню *Папка* и выберете папку *Мои документы*. Ниже будет показано содержимое папки *Мои документы*. Найдите папку с номером вашей группы. Откройте эту папку и найдите файл, соответствующий вашей фамилии, установите на него курсор. Нажмите кнопку *Открыть* и убедитесь, что в верхней левой области экрана после текста *Microsoft Excel* появилось название рабочей папки с вашей фамилией.
2. Перейдите на следующий лист рабочего поля. Если свободные листы отсутствуют, выполните команду *Вставить/Лист*.
3. Последовательностью команд *Сервис/Макрос/Редактор Visual Basic* запустите редактор VBA.
4. Вернитесь к макросу, записанному вами при выполнении предыдущей лабораторной работы.
5. Запустите отладчик и выполните программу по шагам.

6. Введите точки останова и выполните программу с точками останова. Удалите точки останова.
7. Включите окно контрольных значений, введите в него несколько переменных и проконтролируйте в нем изменения значений переменных при пошаговом выполнении программы.
8. Повторите пошаговое выполнение программы, контролируя значения в окне локальных переменных.
9. Воспользуйтесь окном проверка для оперативного изменения текста выполняемой программы, а также для изменения значений переменных.

3. Порядок оформления отчета

Отчетом о лабораторной работе является файл с именем, совпадающим с фамилией студента с результатами работы в папке *Мои документы*/*номер группы*.

4. Контрольные вопросы

1. Перечислите виды ошибок, возникающих в процессе создания и эксплуатации программного обеспечения.
2. Как диагностируются ошибки, выявляемые компилятором VBA? В чем причина этих ошибок?
3. Какие ошибки могут возникнуть на этапе выполнения программы? В чем причина возникновения этих ошибок?
4. Что такое тест и как выполняется тестирование?
5. Каковы виды пошаговых режимов работы отладчика?
6. Какие существуют способы контроля над значениями переменных?
7. Как можно изменить значения переменных в процессе отладки программы?

ЛАБОРАТОРНАЯ РАБОТА №25
Лабораторная работа «Поиск ошибок в программах. Классификация ошибок и тестов

Цель работы: изучить классификацию видов тестирования, практически закрепить эти знания путем генерации тестов различных видов, научиться планировать тестовые активности в зависимости от специфики поставляемой на тестирование функциональности.

Теоретические сведения

Тестирование – процесс, направленный на оценку корректности, полноты и качества разработанного программного обеспечения.

Тестирование можно классифицировать по очень большому количеству признаков. Далее приведен обобщенный список видов тестирования по различным основаниям.

Типы тестов по покрытию (по глубине)

Smoke test – тестирование системы для определения корректной работы базовых функций программы в целом, без углубления в детали. При проведении теста определяется пригодность сборки для дальнейшего тестирования.

Minimal Acceptance Test (MAT, Positive test)*:* тестирование системы или ее части только на валидных данных (валидные данные – это данные, которые необходимо использовать для корректной работы модуля/функции). При тестировании проверяется правильной работы всех функций и модулей с валидными данными.

Для крупных и сложных приложений используется ограниченный набор сценариев и функций.

Acceptance Test (AT): полное тестирование системы или ее части как на корректных, так и на некорректных данных/сценариях. Вид теста, направленный на подтверждение того, что приложение может использоваться по назначению при любых условиях.

Тест на этом уровне покрывает все возможные сценарии тестирования: проверку работоспособности модулей при вводе корректных значений; проверку при вводе некорректных значений; использование форматов данных отличных от тех, которые указаны в требованиях; проверку исключительных ситуаций, сообщений об ошибках; тестирование на различных комбинациях входных параметров; проверку всех классов эквивалентности; тестирование граничных значений интервалов; сценарии не предусмотренные спецификацией и т.д.

Тестовые активности (типы тестов по покрытию (по ширине)):

Defect Validation – проверка результата исправления дефектов.Включает в себя проверку на воспроизводимость дефектов, которые были исправлены в новой сборке продукта, а также проверку того, что исправление не повлияло на ранее работавшую функциональность

New Feature Test (NFT, AT of NF) – определение качества поставленной на тестирование новой функциональности, которая ранее не

тестировалась. Данный тип тестирования включает в себя: проведение полного теста (АТ) непосредственно новой функциональности; тестирование новой функциональности на соответствие документации; проверку всевозможных взаимодействий ранее реализованной функциональности с новыми модулями и функциями.

Regression testing (регрессионное тестирование) **–** проводится с целью оценки качества ранее реализованной функциональности. Включает в себя проверку стабильности ранее реализованной функциональности после внесения изменений, например добавления новой функциональности, исправление дефектов, оптимизация кода, разворачивание приложения на новом окружении. Регрессионное тестирование может быть проведено на уровне Smoke, MAT или AT.

Типы тестов по знанию коду

Черный ящик – тестирование системы, функциональное или нефункциональное, без знания внутренней структуры и компонентов системы. У тестировщика нет доступа к внутренней структуре и коду приложения либо в процессе тестирования он не обращается к ним.

Белый ящик – тестирование основанное на анализе внутренней структуры компонентов или системы. У тестировщика есть доступ к внутренней структуре и коду приложения.

Серый ящик – комбинация методов белого и черного ящика, состоящая в том, что к части кода архитектуры у тестировщика есть, а к части кода – нет.

Типы тестов по степени автоматизации

Ручное – тестирование, в котором тест-кейсы выполняются тестировщиком вручную без использования средств автоматизации.

Автоматизированное – набор техник, подходов и инструментальных средств, позволяющий исключить человека из выполнения некоторых задач в процессе тестирования. Тест-кейсы частично или полностью выполняет специальное инструментальное средство.

Типы тестов по изолированности компонентов

Unit/component (модульное) – тестирование отдельных компонентов (модулей) программного обеспечения.

Integration (интеграционное) – тестируется взаимодействие между интегрированными компонентами или системами.

System (системное) – тестируется работоспособность системы в целом с целью проверки того, что она соответствуе установленным требованиям.

Типы тестов по подготовленности.

Интуитивное тестирование выполняется без подготовки к тестам, без определения ожидаемых результатов, проектирования тестовых сценариев.

Исследовательское тестирование – метод проектирования тестовых сценариев во время выполнения этих сценариев. Тестировщик совершает проверки, продумывает их, придумывает новые проверки, часто использует для этого полученную информацию.

Тестирование по документации – тестирование по подготовленным тестовым сценариям, руководству по осуществлению тестов.

Титы тестов по месту и времени проведения

User Acceptance Testing (UAT) (приемочное тестирование) – формальное тестирование по отношению к потребностям, требованиям и бизнес процессам пользователя, проводимое с целью определения соответствия системы критериям приёмки и дать возможность пользователям, заказчикам или иным авторизованным лицам определить, принимать систему.

Alpha Testing (альфа-тестирование) **–** моделируемое или действительное функциональное тестирование, выполняется в организации, разрабатывающей продукт, но не проектной командой (это может быть независимая команда тестировщиков, потенциальные пользователи, заказчики). Альфа тестирование часто применяется к коробочному программному обеспечению в качестве внутреннего приемочного тестирования.

Beta Testing (бета-тестирование) **–** эксплуатационное тестирование потенциальными или существующими клиентами/заказчиками на внешней стороне (в среде, где продукт будет использоваться) никак связанными с разработчиками, с целью определения действительно ли компонент или система удовлетворяет требованиям клиента/заказчика и вписывается в бизнес-процессы. Бета-тестирование часто проводится как форма внешнего приемочного тестирования готового программного обеспечения для того, чтобы получить отзывы рынка.

Типы тестов по объекту тестирования

Functional testing (функциональное тестирование) – это тестирование, основанное на анализе спецификации, функциональности компонента или системы. Функциональным можно назвать любой вид тестирования, который согласно требованиям проверяет правильную работу.

Safety testing (тестирование безопасности) – тестирование программного продукта с целью определить его безопасность (безопасность

– способность программного продукта при использовании оговоренным образом оставаться в рамках приемлемого риска причинения вреда здоровью, бизнесу, программам, собственности или окружающей среде.

Security testing (тестирование защищенности**)** – это тестирование с целью оценить защищенность программного продукта. Тестирование защищенности проверяет фактическую реакцию защитных механизмов, встроенных в систему, на проникновение.

Compatibility testing (тестирование совместимости) **–** процесс тестирования для определения возможности взаимодействия программного продукта, проверка работоспособности приложения в различных средах (браузеры и их версии, операционные системы, их типа, версии и разрядность)

Виды тестов:

- кросс-браузерное тестирование (различные браузеры или версии браузеров)
- кросс-платформенное тестирование (различные операционные системы или версии операционных систем)

Нефункциональное тестирование – это проверка характеристик программы. Иначе говоря, когда проверяется не именно правильность работы, а какие-либо свойства (внешний вид и удобство пользования, скорость работы и т.п.).

1. Тестирование пользовательского интерфейса (GUI) – тестирование, выполняемое путем взаимодействия с системой через графический интерфейс пользователя.

- навигация
- цвета, графика, оформление
- содержание выводимой информации
- поведение курсора и горячие клавиши
- отображение различного количества данных (нет данных, минимальное и максимальное количество)
- изменение размеров окна или разрешения экрана

2. Тестирование удобства использования (Usability Testing) – тестирование с целью определения степени понятности, легкости в изучении и использовании, привлекательности программного продукта для пользователя при условии использования в заданных условиях эксплуатации.

- визуальное оформление
- навигация
- логичность

3. Тестирование доступности (Accessibility testing) – тестирование, которое определяет степень легкости, с которой пользователи с ограниченными способностями могут использовать систему или ее компоненты.

4. Тестирование интернационализации – тестирование способности продукта работать в локализованных средах (способность изменять элементы интерфейса в зависимости от длины и направления текста, менять сортировки/форматы под различные локали и т.д.). (Максим Черняк).

Интернационализация – это процесс, упрощающий дальнейшую адаптацию продукта к языковым и культурным особенностям региона, отличного от того, в котором разрабатывался продукт. Это адаптация продукта для потенциального использования практически в любом месте, Интернационализация производится на начальных этапах разработки, в то время как локализация — для каждого целевого языка.

5. Тестирование локализации (Localization testing) – тестирование,

проводимое с целью проверить качество перевода продукта с одного языка на другой.

6. Тестирование производительности или нагрузочное тестирование **–** процесс тестирования с целью определения производительности программного продукта.

Виды тестов:

- нагрузочное тестирование (Performance and Load testing) – вид тестирования производительности, проводимый с целью оценки поведения компонента или системы при возрастающей нагрузке, например количестве параллельных пользователей и/или операций, а также определения какую нагрузку может выдержать компонент или система;
- объемное тестирование (Volume testing) – позволяет получить оценку производительности при увеличении объемов данных в базе данных приложения;
- тестирование стабильности и надежности (Stability / Reliability testing) – позволяет проверять работоспособность приложения при длительном (многочасовом) тестировании со средним уровнем нагрузки.
- стрессовое тестирование (Stress testing) – вид тестирования производительности, оценивающий систему или компонент на граничных значениях рабочих нагрузок или за их пределами, или же в состоянии ограниченных ресурсов, таких как память или доступ к серверу.

7. Тестирование требований (Requirements testing) – проверка требований на соответствие основным характеристикам качества.

8. Тестирование прототипа (Prototyte testing) – метод выявления структурных, логических ошибок и ошибок проектирования на ранней стадии развития продукта до начала фактической разработки.

9. Тестирование установки (Installability testing) и лицензирования – процесс тестирования устанавливаемости программного продукта.

Виды тестов:

- формальный тест программы установки приложения (проверка пользовательского интерфейса, навигации, удобства пользования, соответствия общепринятым стандартам оформления);
- функциональный тест программы установки;
- тестирование механизма лицензирования и функций защиты от пиратства;
- проверка стабильности приложения после установки.

10. ***Тестирование на отказ и восстановление (Failover and Recovery Testing)*** – тестирование при помощи эмуляции отказов системы или реально вызываемых отказов в управляемом окружении.

11. **Тестирование программного продукта включает следующие этапы:**

1. Изучение и анализ предмета тестирования.
2. Планирование тестирования.
3. Исполнение тестирования.

Изучение и анализ предмета тестирования начинается еще до утверждения спецификации и продолжается на стадии разработки (кодирования) программного обеспечения. Конечной целью этапа изучение и анализ предмета тестирования является получение ответов на два вопроса:

- какие функциональности предстоит протестировать,
- как эти функциональности работают.

Планирование тестирования происходит на стадии разработки (кодирования) программного обеспечения. На стадии планирования тестирования перед тестировщиком стоит задача поиска компромисса между объемом тестирования, который возможен в теории, и объемом тестирования, который возможен на практике. На данной стадии необходимо ответить на вопрос: как будем тестировать? Результатом планирования тестирования является тестовая документация.

Выполнение тестирования происходит на стадии тестирования и представляет собой практический поиск дефектов с использованием тестовой документации, составленной ранее.

Для всех программных продуктов выполняют следующие типы тестов и их композиции.

Для первого билда рекомендуется проводить Smoke+AT готовой функциональности: поверхностное тестирование (Smoke Test) выполняется для определения пригодности сборки для дальнейшего тестирования; полное тестирование системы или ее части как на корректных, так и на некорректных данных/сценариях (Acceptance Test, AT) позволяет обнаружить дефекты и внести запись о них в багтрэкинговую систему.

Для последующих билдов композиции тестов могут быть следующими:

- Если не была добавлена новая функциональность, то: DV+MAT. Т.е., выполняется проверка исправления дефектов программистом (Defect Validation, DV), а также проверка работоспособности остальной функциональности после исправления дефектов на позитивных сценариях (Minimal Acceptance Test, MAT).
- Если была добавлена новая функциональность, то: Smoke+DV+NFT+Regression Test. В частности, выполняется поверхностное тестирование (Smoke Test), проверка исправления дефектов программистом (Defect Validation, DV), тестирование новых функциональностей (New Feature Testing, NFT), проверка старых функциональностей, т.е. регрессионное тестирование (Regression Test).
- Если была добавлена новая функциональность, то возможен также вариант: DV+NFT+Resression test, т.е. без выполнения Smoke Test.

В зависимости от типа и специфики приложения (web, desktop, mobile) выполняют специализированные тесты (например, кроссбраузерное или кроссплатформенное тестирование, тестирование локализации и интернационализации и др.).

Порядок выполнения работы

1. Получить задание у преподавателя.

2. Выполнить генерацию тестов различных видов для конкретного объекта реального мира (пример приведен на рисунке 1).

3. Спланировать тестовые активности для следующих задач:

3.1 Поставлен на тестирование модуль 1, модуль 2, модуль 3.

3.2 Проведены исправления (fix) для заведенных дефектов, доставлена новая функциональность – модуль 4.

3.3 Заказчик решил расширять рынки сбыта и просит осуществить поддержку для Великобритании (кроме уже существующей Беларуси).

3.4 Заказчик хочет убедиться, что ПО держит нагрузку в 2000 пользователей.

4. Оформить отчет и защитить лабораторную работу.

Пример выполнения лабораторной работы

Необходимо составить тестовый план для объекта «Карандаш».

Пример тестового плана для объекта карандаш представлен на рисунке 1.1.

Рисунок 1.1 – Пример генерации тестов различных видов для объекта «Карандаш»*одержание отчета*

1. Цель работы.
2. Краткие теоретические сведения.
3. Сгенерированные тесты различных видов для выбранного объекта реального мира.
4. Тестовые активности для сформулированных задач.
5. Выводы по работе.

Контрольные вопросы

1. Что такое тестирование?
2. Какие существуют типы тестов по покрытию? Дайте характеристику каждому.
3. Какие существуют тестовые активности? Дайте характеристику каждому.
4. Какие существуют типы тестов знанию кода? Дайте характеристику каждому.
5. Какие существуют типы тестов по степени автоматизации? Дайте характеристику каждому.
6. Какие существуют типы тестов по изолированности компонентов? Дайте характеристику каждому.
7. Какие существуют типы тестов по подготовленности? Дайте характеристику каждому.
8. Какие существуют типы тестов по месту и времени проведения? Дайте характеристику каждому.
9. Какие существуют типы тестов по объекту тестирования? Дайте характеристику каждому.

10. Какие существуют типы функциональных тестов? Дайте характеристику каждому.

11. Какие существуют типы нефункциональных тестов? Дайте характеристику каждому.

12. Какие этапы составляют процесс тестирования?

13. Что происходит на этапе изучения и анализа предмета тестирования?

14. Что происходит на этапе планирования тестирования?

15. Что происходит на этапе исполнения тестирования?

16. Какие типы тестов выполняют для первой поставки программного продукта?

17. Какие типы тестов выполняют для последующих поставок программного продукта?

Лабораторная работа №26-27

Лабораторная работа «Моделирование бизнес-процессов в ИС»

1. **Цель работы** Ознакомление студентов с методологией функционального моделирования бизнес-процессов. Приобретение навыков создания и редактирования функциональных моделей в BPwin.

2. **Содержание работы** Изучить методы, правила и процедуры функционального моделирования (методологию SADT/IDEF0). Приобрести навыки построения модели с помощью инструментального CASE-средства BPWin на примере структурного анализа деятельности некоторой торговой фирмы.

3. **Теоретические сведения**

Постоянное усложнение производственно-технических и организационно-экономических систем, поиск возможностей повышения их эффективности обусловливают необходимость применения специальных средств описания и анализа сложных систем.

Важнейшими характеристиками любой системы являются её структура и процесс функционирования. Под структурой системы понимают устойчивую во времени совокупность взаимосвязей между её элементами или компонентами. Именно структура связывает воедино все элементы и препятствует распаду системы на отдельные компоненты.

Процесс функционирования системы тесно связан с изменением её свойств или поведения во времени. При этом важной характеристикой системы является её состояние - совокупность свойств или признаков, которые в каждый момент времени отражают наиболее существенные особенности поведения системы.

Бизнес-система – это такая организация среды, которая имеет внутренние и внешние границы, взаимодействует с внутренними элементами и внешними агентами путем обмена деловой информацией через имеющиеся каналы связи, используя принятые правила, а, также, обладая сложной организационной иерархической структурой, осуществляет экономическую деятельность с главной целью получения прибыли и саморазвития.

Методы моделирования бизнес-процессов

Для моделирования бизнес-процессов используется несколько различных методов, основой которых являются как структурный, так и объектно-ориентированный подходы к моделированию. К числу наиболее распространенных методов относятся:

- метод функционального моделирования SADT (IDEF0);
- метод моделирования процессов IDEF3;
- моделирование потоков данных DFD;

- метод ARIS; метод Ericsson-Penker;
- метод моделирования, используемый в технологии Rational Unified Process.

Метод функционального моделирования SADT (IDEF0)

Метод SADT (Structured Analysis and Design Technique) считается классическим методом процессного подхода к управлению. Основной принцип процессного подхода заключается в структурировании деятельности организации в соответствии с ее бизнес-процессами, а не организационно-штатной структурой. Именно бизнес-процессы, формирующие значимый для потребителя результат, представляют ценность, и именно их улучшением предстоит в дальнейшем заниматься. Модель, основанная на организационно-штатной структуре, может продемонстрировать лишь хаос, царящий в организации, на ее основе можно только внести предложения об изменении этой структуры. С другой стороны, модель, основанная на бизнес-процессах, содержит в себе и организационно-штатную структуру предприятия.

В соответствии с этим принципом бизнес-модель должна выглядеть следующим образом:

1. Верхний уровень модели должен отражать только контекст системы — взаимодействие моделируемого единственным контекстным процессом предприятия с внешним миром.
2. На втором уровне модели должны быть отражены основные виды деятельности (тематически сгруппированные бизнес-процессы) предприятия и их взаимосвязи. В случае большого их количества некоторые из них можно вынести на третий уровень модели. Но в любом случае под виды деятельности необходимо отводить не более двух уровней модели.
3. Дальнейшая детализация бизнес-процессов осуществляется посредством бизнес-функций — совокупностей операций, сгруппированных по определенным признакам. Бизнес-функции детализируются с помощью элементарных бизнес-операций.
4. Описание элементарной бизнес-операции осуществляется посредством задания алгоритма ее выполнения.

Метод SADT разработан Дугласом Россом в 1969 г. для моделирования искусственных систем средней сложности.

Технология SADT успешно использовалась в военных, промышленных и коммерческих организациях США для решения широкого круга задач, таких как долгосрочное и стратегическое планирование, автоматизированное производство и проектирование, разработка ПО для оборонных систем, управление финансами и материально-техническим снабжением и др. Метод SADT поддерживается Министерством обороны США, которое было

инициатором разработки семейства стандартов IDEF (Icam DEFinition), являющегося основной частью программы ICAM (интегрированная компьютеризация производства), проводимой по инициативе ВВС США. Метод SADT реализован в одном из стандартов этого семейства — IDEF0, который был утвержден в качестве федерального стандарта США в 1993 г. Существует также российская версия данного стандарта.

Метод SADT представляет собой совокупность правил и процедур, предназначенных для построения функциональной модели объекта какой-либо предметной области. Функциональная модель SADT отображает функциональную структуру объекта, т.е. производимые им действия и связи между этими действиями. Основные элементы этого метода основываются на следующих концепциях:

1. ***Графическое представление блочного моделирования***. Графика блоков и дуг SADT-диаграммы отображает функцию в виде блока, а интерфейсы входа/выхода представляются дугами, соответственно входящими в блок и выходящими из него. Взаимодействие блоков друг с другом описывается посредством интерфейсных дуг, выражающих "ограничения", которые, в свою очередь, определяют, когда и каким образом функции выполняются и управляются.

2. ***Строгость и точность***. Выполнение правил SADT требует достаточной строгости и точности. Правила SADT включают: ограничение количества блоков на каждом уровне декомпозиции (правило 3-6 блоков — ограничение мощности краткосрочной памяти человека), связность диаграмм (номера блоков), уникальность меток и наименований (отсутствие повторяющихся имен), синтаксические правила для графики (блоков и дуг), разделение входов и управлений (правило определения роли данных).

3. ***Отделение*** *организации от функции*, т.е. исключение влияния административной структуры организации на функциональную модель.

Метод SADT может использоваться для моделирования самых разнообразных процессов и систем. Для новых систем применение метода имеет своей целью определение требований и указание функций для последующей разработки системы, отвечающей поставленным требованиям и реализующей выделенные функции. В существующих системах метод SADT может быть использован для анализа функций, выполняемых системой, и указания механизмов, посредством которых они осуществляются.

Результатом применения метода SADT является модель, которая состоит из диаграмм, фрагментов текстов и глоссария, имеющих ссылки друг на друга.

Диаграммы — главные компоненты модели. Двумя наиболее важными компонентами диаграмм являются бизнес функции или работы (представленные на диаграммах в виде прямоугольников) и данные или

объекты (изображаемые в виде стрелок), связывающие между собой работы. При этом стрелки, в зависимости от того в какую грань прямоугольника работы они входят или из какой грани выходят, делятся на пять видов:

- стрелки входа (входят в левую грань работы) — изображают данные или объекты, изменяемые в ходе выполнения работы.
- стрелки управления (входят в верхнюю грань работы) — изображают правила и ограничения, согласно которым выполняется работа.
- стрелки выхода (выходят из правой грани работы) — изображают данные или объекты, появляющиеся в результате выполнения работы.
- стрелки механизма (входят в нижнюю грань работы) — изображают ресурсы, необходимые для выполнения работы, но не изменяющиеся в процессе работы (например, оборудование, людские ресурсы…)
- стрелки вызова (выходят из нижней грани работы) — изображают связи между разными диаграммами или моделями, указывая на некоторую диаграмму, где данная работа рассмотрена более подробно.

Все работы и стрелки должны быть именованы. (рис. 3.1.).

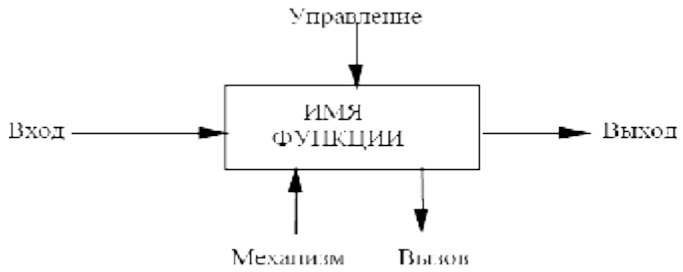


Рис. 3.1. - Функциональный блок и интерфейсные дуги

Одной из наиболее важных особенностей метода SADT является постепенное введение все больших уровней детализации по мере создания диаграмм, отображающих модель. Каждый компонент модели может быть декомпозирован на другой диаграмме. Каждая диаграмма иллюстрирует "внутреннее строение" блока на родительской диаграмме.

Для автоматизации процесса построения диаграмм моделей используются программно - технологические средства, которые получили название CASE-средств, реализующие CASE – технологию создания и сопровождения информационных систем.

Термин CASE (Computer-Aided Software/System Engineering) дословно переводится как разработка программного обеспечения с помощью компьютера.

CASE-средствапредставляют собой программные средства, поддерживающие процессы создания и/или сопровождения информационных систем.

CASE-технология представляет собой совокупность методологий анализа, проектирования, разработки и сопровождения сложных систем программного обеспечения (ПО), поддержанную комплексом взаимоувязаемых средств автоматизации. CASE – это инструментарий для системных аналитиков, разработчиков и программистов, заменяющий им бумагу и карандаш

В большинстве современных CASE-систем применяются методологии структурного анализа и проектирования, основанные на наглядных диаграммных техниках. Такие методологии обеспечивают строгое и наглядное описание проектируемой системы, которое начинается с ее общего обзора и затем детализируется, приобретая иерархическую структуру со все большим числом уровней.

Создание модели процессов в BPWin

CASE-средство фирмы PLATINUM technology BPWin представляет собой средство для сбора всей необходимой информации о некоторой системе и графического изображения этой информации в виде целостной и непротиворечивой модели.

Методы моделирования в BPWin

BPWin поддерживает три методологии: IDEF0, DFD и IDEF3, позволяющие анализировать систему с трех ключевых точек зрения:

1. ***С точки зрения функциональности системы***. В рамках методологии IDEF0 (Integration Definition for Function Modeling) бизнес-процесс представляется в виде набора элементов-работ, которые взаимодействуют между собой, а также показывается информационные, людские и производственные ресурсы, потребляемые каждой работой.
2. ***С точки зрения потоков информации*** (документооборота) в системе. Диаграммы DFD (Data Flow Diagramming) могут дополнить то, что уже отражено в модели IDEF0, поскольку они описывают потоки данных, позволяя проследить, каким образом происходит обмен информацией между бизнес функциями внутри системы. В тоже время диаграммы DFD оставляют без внимания взаимодействие между бизнес функциями.
3. ***С точки зрения последовательности выполняемых работ***. Еще более точную картину можно получить, дополнив модель диаграммами IDEF3. Этот метод привлекает внимание к очередности выполнения событий. В IDEF3 включены элементы логики, что позволяет моделировать и анализировать альтернативные сценарии развития бизнес процесса.

Основной из трех методологий, поддерживаемых BPWin , является IDEF0. Как уже отмечалось, IDEF0 относится к семейству стандартов IDEF, которое широко применяется при разработке моделей бизнес-систем.

Результатом применения IDEF0 к некоторой системе является модель этой системы, состоящая из иерархически упорядоченного набора диаграмм,

текста документации и словарей, связанных друг с другом с помощью перекрестных ссылок.

Модель может содержать следующие типы диаграмм: контекстную диаграмму, диаграммы декомпозиции, диаграмму дерева узлов.

Контекстная диаграмма в модели может быть только одна. Она является вершиной древовидной структуры диаграмм и представляет собой самое общее описание системы и её взаимодействия с внешней средой.

После описания системы в целом проводится её разбиение на крупные фрагменты. Этот процесс называется функциональной декомпозицией, а диаграммы, которые описывают каждый фрагмент и взаимодействие фрагментов, называются диаграммами декомпозиции.

Диаграмма дерева узлов показывает иерархическую зависимость работ, не отражая взаимосвязи между работами.

Процесс моделирования любой системы в IDEF0 начинается с определения контекста, т.е. наиболее абстрактного уровня описания системы в целом. В контекст входит определение субъекта моделирования, цели и точки зрения на модель.

Субъект — это сама система, при этом необходимо точно установить, что входит в систему, а что лежит за её пределами. Иными словами, первоначально следует определить область моделирования. При описании области необходимо учитывать два компонента – широту и глубину. Широта определяет границы модели. Глубина определяет, на каком уровне детализации модель является завершённой.

При формулировании цели следует ответить на следующие вопросы:

1. Почему нужно построить модель процесса?
2. Что должна показывать модель?
3. Что может получить эксперт?

Точку зрения можно представить как взгляд человека, который видит систему в нужном для моделирования направлении. Точка зрения должна соответствовать цели моделирования. Как правило, выбирается точка зрения человека, ответственного за моделируемую работу в целом.

BPWin умеет проверять создаваемые модели с точки зрения синтаксиса выбранной методологии, проверяет ссылочную целостность между диаграммами, а также выполняет ряд других проверок, чтобы помочь вам создать правильную модель, а не просто рисунок. При этом сохраняются главные преимущества рисунка — простота создания и наглядность.

BPWin тесно интегрируется с такими программными продуктами, как:

- система моделирования данных ERwin (Platinum Technology Inc.);
- комплекс управления и хранения проектов ModelMart (Platinum Technology Inc.);

- специализированный генератор отчетов по модели RPTwin (Platinum Technology Inc.);
- система имитационного моделирования BPSimulator (System Modeling Corporation);
- инструмент стоимостного анализа EasyABC (ABC Technologies).

Интерфейс BPWin

BPWin имеет достаточно простой и интуитивно понятный интерфейс пользователя, дающий возможность аналитику создавать сложные модели при минимальных усилиях. Рассмотрим особенности интерфейса BPWin версии 4.0.

В BPWin возможно построение смешанных моделей, т.е. модель может содержать одновременно как диаграммы IDEF0, так и IDEF3 и DFD. Состав палитры инструментов в окне BPWin изменяется автоматически тогда, когда происходит переключение с одной нотации на другую.

Рабочее окно программыBPWin (рис. 3.2.) содержит главное меню программы, основную панель инструментов, палитру инструментов (рис. 3.3.), навигатор модели (Model Explorer) и область построения диаграммы.

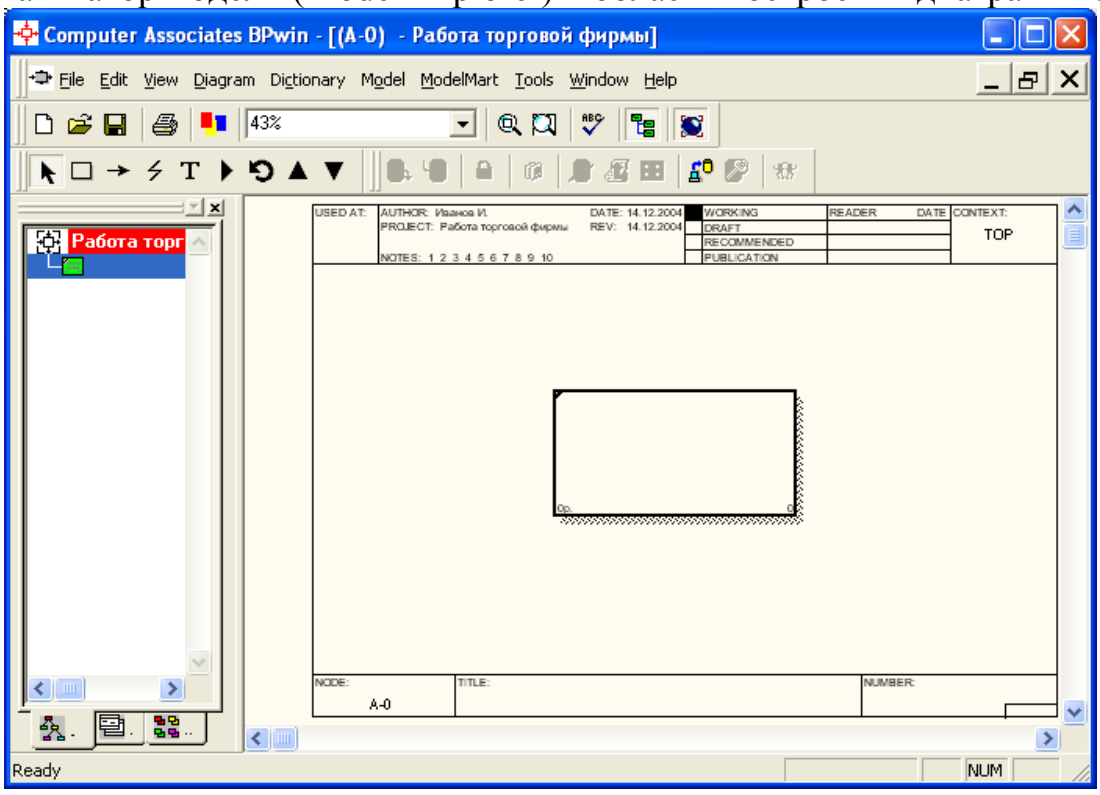


Рис. 3.2 - Интегрированная среда разработки модели BPWin 4.1

Панель инструментов для любой методологии содержит основные элементы управления, представленные в таблице 3.1. В таблице также приведены соответствующие пункты меню, которые позволяют выполнить эти операции. Для построения диаграмм в нотации IDEF0 используются

элементы управления палитры инструментов, представленные на рис. 3.3. Табл. 3.1 - Описание элементов управления основной панели инструментов BPWin

Элемент управления	Описание	Соответствующий пункт меню
	Создание новой модели	File/New
	Открытие модели	File/Open
	Сохранение модели	File/Save
	Печать модели	File/Print
	Создание отчёта	Tools/Report Bulder
	Выбор масштаба	View/Zoom
	Масштабирование	View/Zoom
	Проверка правописания	Tools/Spelling
	Включение и выключение навигатора модели Model Explorer	View/Model Explorer
	Включение и выключение дополнительной панели инструментов работы с ModelMart	ModelMart
	- редактирование объектов - добавление работы на диаграмму - проведение новой связи - связь стрелки с подписью - добавление текстового блока - открытие окна редактора диаграмм - переход между стандартной диаграммой и деревом узлов - переход на родительскую диаграмму -декомозиция диаграмм	

Рис. 3.3. - Палитра инструментов нотации IDEF0

Установка цвета и шрифта объектов

Цвет и шрифт (в том числе размер и стиль) объектов можно установить с помощью команд контекстного меню Font Editor и Color Editor. Кроме того, BPWin позволяет установить шрифт по умолчанию для объектов определенного типа на диаграммах и в отчетах. Для этого следует выбрать в меню **Tools** команду **Default Fonts**, после чего появляется каскадное меню,

каждый пункт которого служит для установки шрифта для определенного типа объектов:

Context Activity - работа на контекстной диаграмме;

Context Arrow - стрелки на контекстной диаграмме;

Decomposition Activity - работы на диаграмме декомпозиции;

Decomposition Arrow - стрелки на диаграмме декомпозиции;

NodeTree Text - текст на диаграмме дерева узлов;

Frame User Text - текст, вносимый пользователем в каркасе диаграмм;

Frame System Text; - системный текст в каркасе диаграмм;

Text Blocks - текстовые блоки;

Parent Diagram Text - текст родительской диаграммы;

Parent Diagram Title Text - текст заголовка родительской диаграммы;

Report Text - текст отчетов.

Построение диаграмм

Бизнес-процесс в BPWin в соответствии с методологией IDEFO описывается в виде совокупности иерархически упорядоченных и взаимосвязанных диаграмм (контекстной, диаграмм декомпозиции и диаграммы дерева узлов). Каждая диаграмма является единицей описания системы и располагается на отдельном листе.

Модель в BPWin рассматривается как совокупность работ, каждая из которых оперирует с некоторым набором данных. Работа изображается в виде прямоугольников, данные - в виде стрелок. Если щелкнуть по любому объекту модели левой кнопкой мыши, появляется всплывающее контекстное меню, каждый пункт которого соответствует редактору какого-либо свойства объекта.

Стрелки на контекстной диаграмме служат для описания взаимодействия системы с окружающим миром. Они могут начинаться у границы диаграммы и заканчиваться у работы, или наоборот. Такие стрелки называются граничными.

Рис. 3.4. Пример контекстной диаграммы

Имена вновь внесенных стрелок автоматически заносятся в словарь “Arrow Dictionary”. Словарь стрелок решает очень важную задачу. Диаграммы создаются аналитиком для того, чтобы провести сеанс экспертизы, т. е. обсудить диаграмму со специалистом предметной области. В любой предметной области формируется профессиональный жаргон, причем очень часто жаргонные выражения имеют нечеткий смысл и воспринимаются разными специалистами по-разному. В то же время аналитик - автор диаграмм должен употреблять те выражения, которые наиболее понятны экспертам. Поскольку формальные определения часто сложны для восприятия, аналитик вынужден употреблять профессиональный жаргон, а, чтобы не возникло неоднозначных трактовок, в словаре стрелок каждому понятию можно дать расширенное и, если это необходимо, формальное определение.

Диаграмма декомпозиции предназначена для детализации работы. В отличие от моделей, отображающих структуру организации, работа на диаграмме верхнего уровня в IDEF0 - это не элемент управления нижестоящими работами. Работы нижнего уровня - это то же самое, что работы верхнего уровня, но в более детальном изложении. Как следствие этого границы работы верхнего уровня - это то же самое, что границы диаграммы декомпозиции.

4. Постановка задачи

В лабораторной работе необходимо выполнить системный анализ заданной предметной области и построить функциональную модель бизнес-процессов некоторой организациию при помощи SADT-диаграмм. Создание модели необходимо выполнить с применением CASE-средства BPWin.

Модель должна содержать контекстную диаграмму и диаграмму декомпозиции 1-го уровня, согласно методологии IDEF0, а также созданный в BPwin отчет по модели.

5. Требования к оформлению лабораторной работы

1. Создать в среде Word отчёт, содержащий описание предметной области.

2. Привести в отчёте и коротко описать все созданные диаграммы, а также созданный в BPwin отчет по модели.

3. Продемонстрировать разработанную модель в электронном виде (*.bp1).

6. Варианты заданий

1. Ателье мод.
2. Химчистка.
3. Банк.
4. Агентство недвижимости.
5. Бассейн.
6. Продажа автомобилей.
7. Благотворительная организация «Милосердие».
8. Охранное агентство.
9. Выставочный зал.
10. Типография.
11. Спутниковое телевидение.
12. Кафедра ВУЗа.
13. Центр занятости.
14. Установка пластиковых окон.
15. Кинотеатр.

7. Пример выполнения лабораторной работы

В качестве примера рассматривается деятельность вымышленной компании «**Computer Word**». Компания занимается в основном сборкой и продажей настольных компьютеров и ноутбуков. Компания не производит компоненты самостоятельно, а только собирает и тестирует компьютеры.

Основные виды работ в компании таковы:

- продавцы принимают заказы клиентов;
- операторы группируют заказы по типам компьютеров;

- опера

торы собирают и тестируют компьютеры;

- опера

торы упаковывают компьютеры согласно заказам;

- кладо

вщик отгружает клиентам заказы.

Компания использует лицензионную бухгалтерскую информационную систему, которая позволяет оформить заказ, счет и отследить платежи по счетам.

1. Запус

тите **BPwin**. (Кнопка Start 

/BPwin 

).

2. Если

появляется диалог **ModelMart Connection Manager**, нажмите на кнопку **Cancel** (Отмена).

3. Щелк

ните по кнопке  . Появляется диалоговое окно **I would like to** (рисунок 7.1). Внесите в текстовое поле **Name** имя модели "Деятельность компании" и выберите **Type – Business Process (IDEF0)**. Нажмите кнопку **ОК**.

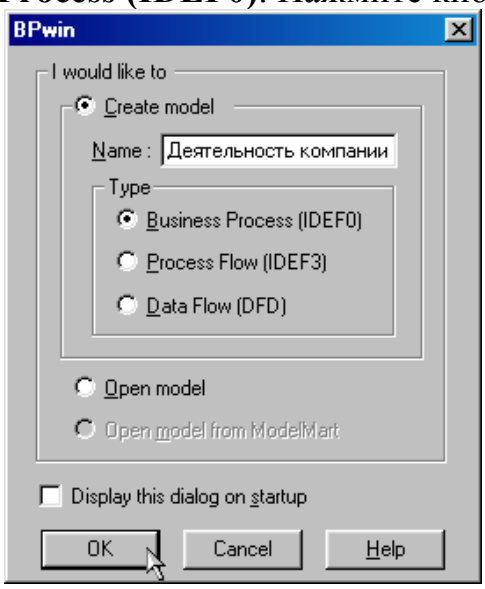


Рисунок 7.1 – Присвоение модели имени и выбор типа модели

4. Откр

оется диалоговое окно **Properties for New Models (**Свойства новой модели**)** (рисунок 7.2).

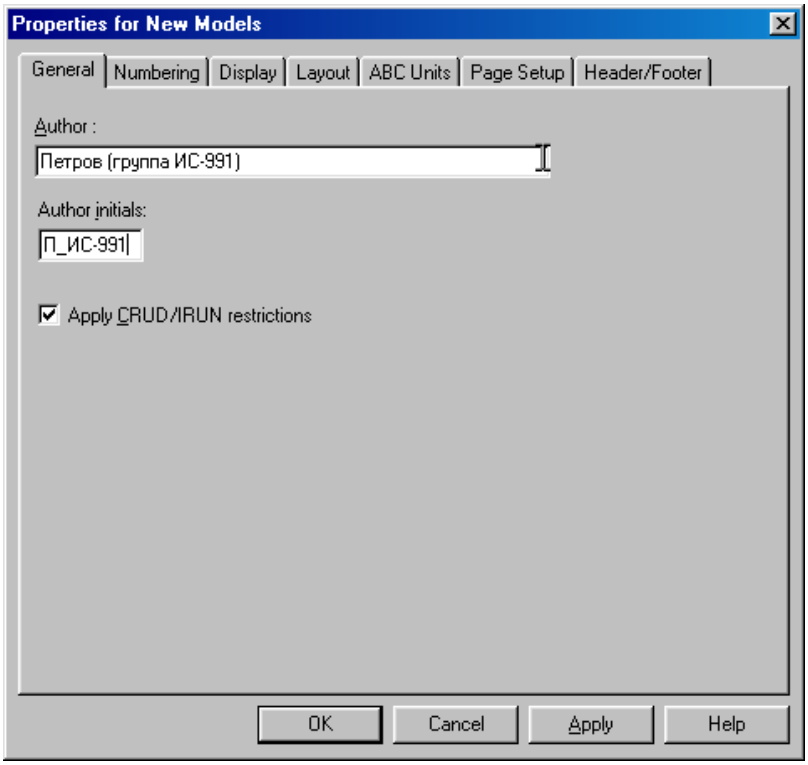


Рисунок 7.2 - Ввод имени автора модели и его инициалов

Введите в текстовое поле **Author** (Автор) имя автора модели и в текстовое поле **Author initials** его инициалы. Нажмите последовательно кнопки **Apply** и **ОК**.

5. Автоматически создается незаполненная контекстная диаграмма (рисунок 7.3).

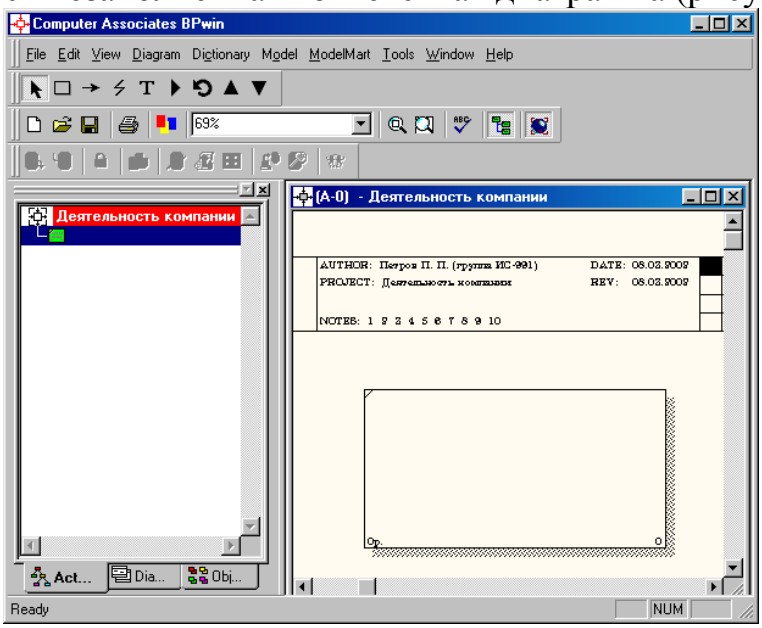
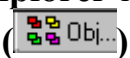


Рисунок 7.3 – Незаполненная контекстная диаграмма

6. Обратите внимание на кнопку  на панели инструментов. Эта кнопка включает и выключает инструмент просмотра и навигации - **Model Explorer** (Браузер модели). **Model Explorer** имеет три вкладки –**Activities** (

), **Diagrams** (

) и **Objects** (

). Во вкладке **Activities** щелчок правой кнопкой по

объекту в браузере модели позволяет выбрать опции редактирования его свойств (рисунок 7.4).

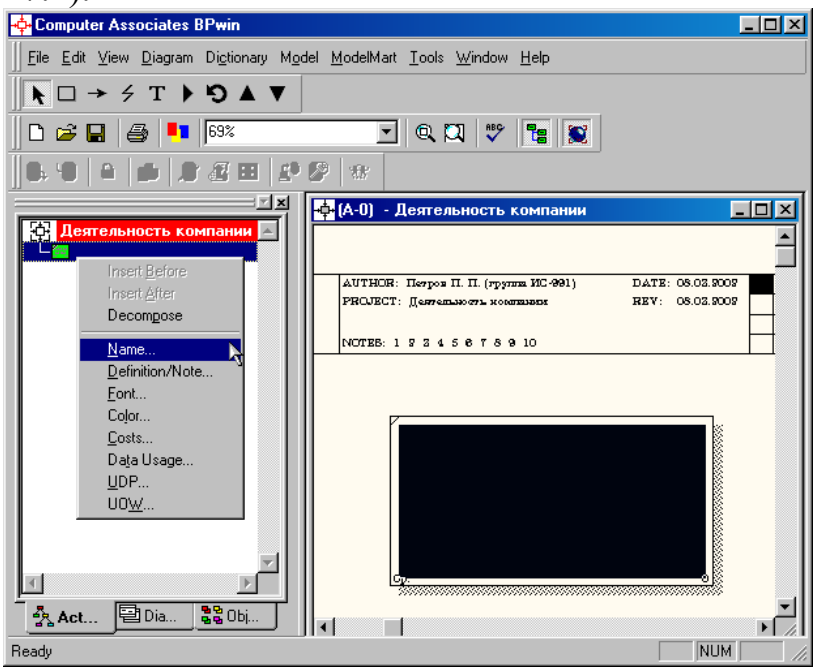


Рисунок 7.4 – Щелчок правой кнопкой по объекту во вкладке **Activities** позволяет воспользоваться контекстным меню для редактирования его свойств

7. Если вам непонятно, как выполнить то или иное действие, вы можете вызвать контекстную помощь - клавиша **F1** или воспользоваться меню **Help**.

8. Перейдите в меню **Model/Model Properties**. Во вкладке **General** диалогового окна **Model Properties** в текстовое поле **Model name** следует внести имя модели "Деятельность компании", а в текстовое поле **Project** имя проекта "Модель деятельности компании", и, наконец, в текстовое **Time Frame** (Временной охват) - **AS-IS (**Как есть**)** (рисунок 7.5).

9. Во вкладке **Purpose** диалогового окна **Model Properties** в текстовое поле **Purpose** (цель) внесите данные о цели разработки модели - " Моделировать текущие (AS-IS) бизнес-процессы компании", а в текстовое поле **Viewpoint** (точка зрения) - "Директор".

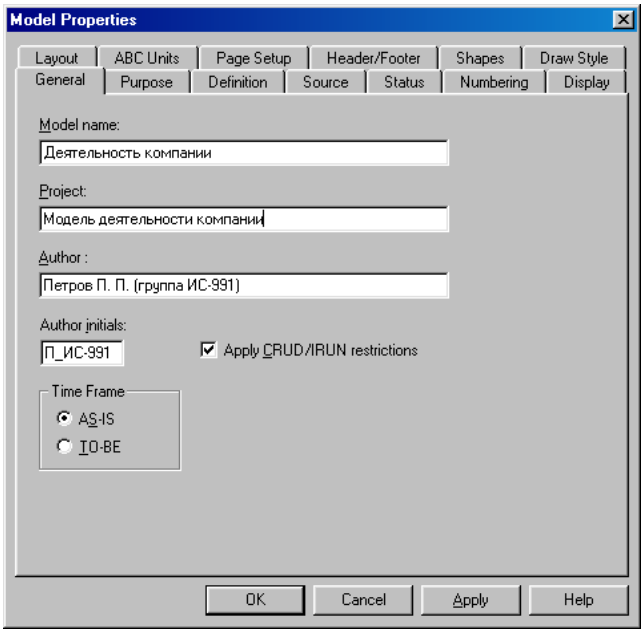


Рисунок 7.5 – Окно задания свойств модели

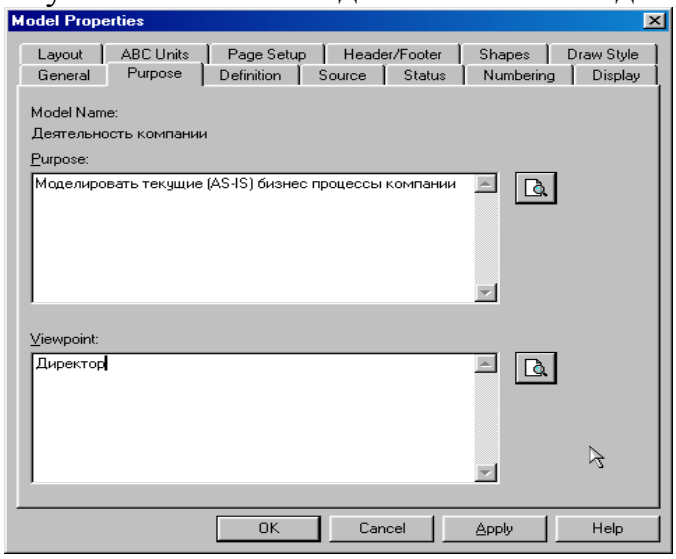


Рисунок 7.6 – Внесение данных о цели моделирования и точке зрения на модель

10. Во вкладке Definition диалогового окна Model Properties в текстовое поле Definition (Определение) внесите "Это учебная модель, описывающая деятельность компании" и в текстовое поле Scope (охват) - " Общее управление бизнесом компании: исследование рынка, закупка компонентов, сборка, тестирование и продажа продуктов".

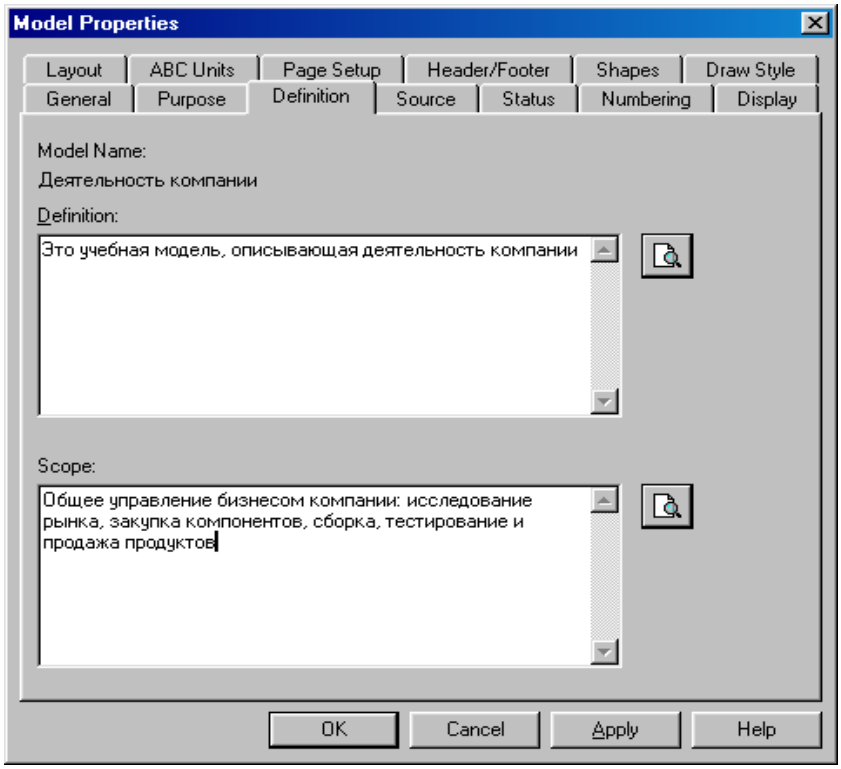


Рисунок 7.7 – Внесение дополнительных данных определяющих модель

11. Перейдите на контекстную диаграмму и правой кнопкой мыши щелкните по прямоугольнику представляющему, в нотации IDEF0, условное графическое обозначение работы. В контекстном меню выберите опцию Name (рисунок 7.8). Во вкладке Name внесите имя "Деятельность компании" (рисунок 7.9).

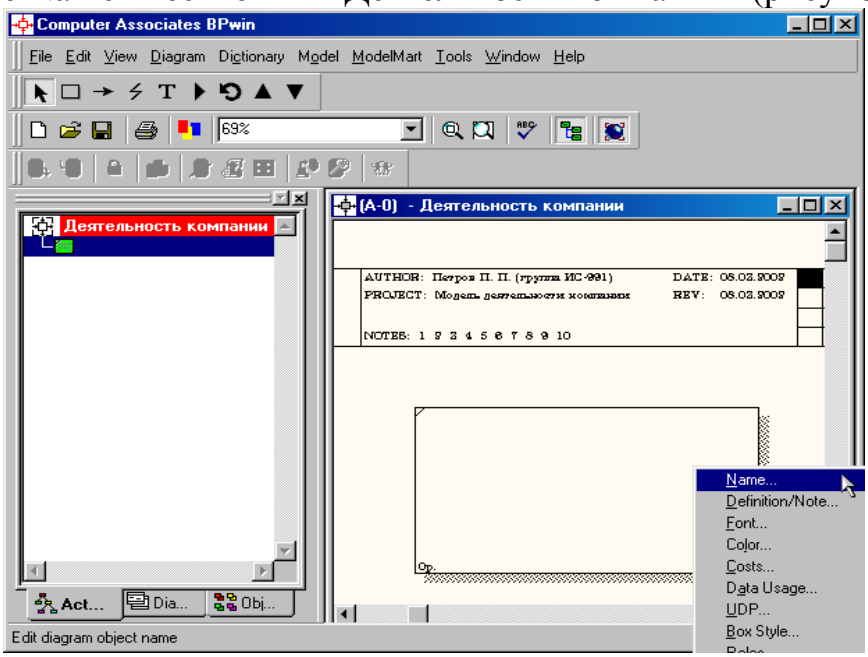


Рисунок 7.8 – Контекстное меню для работы с выбранной опцией **Name**

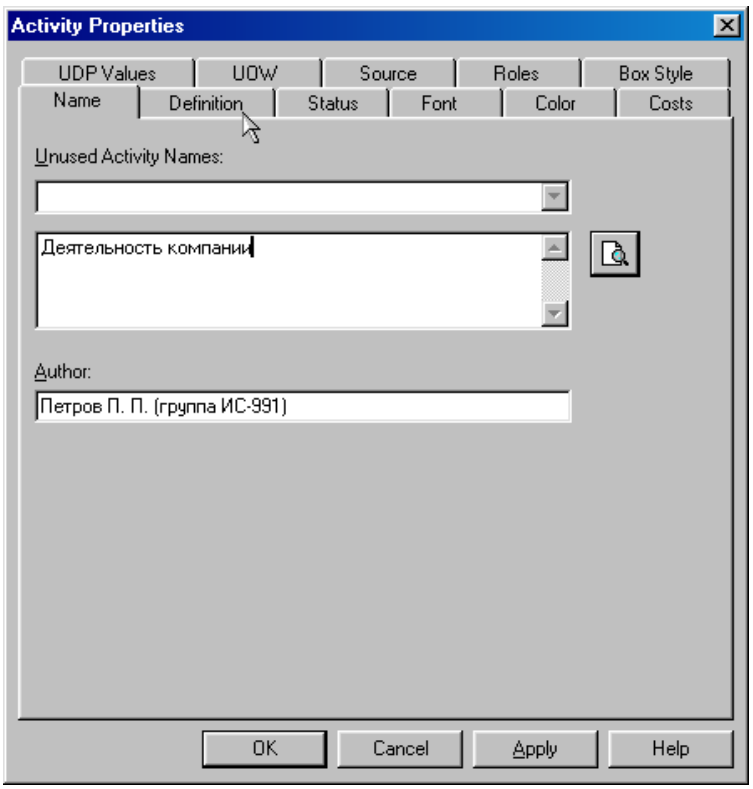


Рисунок 7.9 – Присвоение работе названия

12. Во вкладке Definition диалогового окна Activity Properties в текстовое поле Definition (Определение) внесите "Текущие бизнес-процессы компании" (рисунок 7.10). Текстовое поле Note (Примечания) оставьте незаполненным.

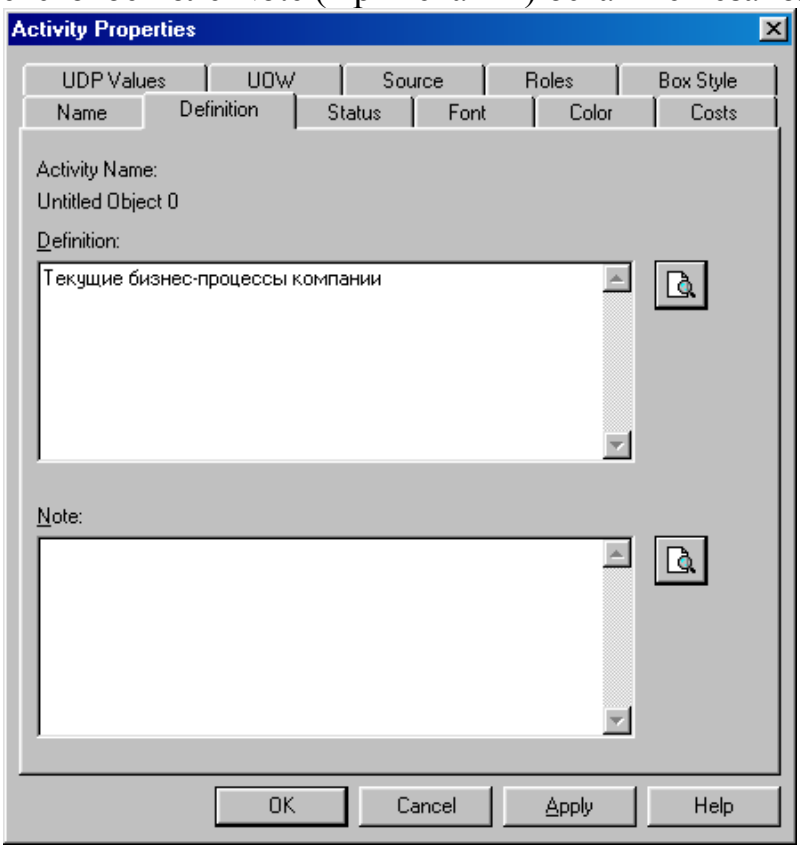


Рисунок 7.10 – Внесение дополнительных данных о работе

13. Создайте ICOM-стрелки на контекстной диаграмме (таблица 7.1).

Стрелки на контекстной диаграмме служат для описания взаимодействия системы с окружающим миром. Они могут начинаться у границы диаграммы и заканчиваться у работы, или наоборот. Такие стрелки называются граничными.

Для внесения граничных стрелок входа, управления и механизма выполните следующие действия:

- Выберите кнопку (проведение новой связи) на палитре инструментов.
- Подведите курсор к соответствующей границе рабочей области диаграммы. На границе должна появиться начальная штриховая полоска.

- Щелкните один раз по полоске, а затем по той стороне работы, куда входит стрелка. При наведении указателя на сторону работы, с внутренней стороны работы появляется закрашенный треугольник.
- Вернитесь на палитру инструментов и выберите кнопку редактирования объектов.
- Вызовите контекстное меню, в котором выберите пункт именования стрелки **Name**, в открывшемся окне “Arrow Properties” в закладке **Name** введите имя стрелки.

Для стрелок выхода последовательность действий изменяется. Вначале щелкните на стороне работы, а затем на штриховке границе

Введённое имя стрелки можно растянуть в одну строку или сжать, расположив слова имени вертикально одно под другим. Для этого в режиме редактирования выделите левой кнопкой мышки текст имени и не отпуская её, растяните или сожмите границы текста. На рис. 7.11. приведен пример контекстной диаграммы.

Таблица 7.1 - Стрелки контекстной диаграммы

Название стрелки (Arrow Name)	**Определение стрелки (Arrow Definition)**	**Тип стрелки (Arrow Type)**
Звонки клиентов	Запросы информации, заказы, техподдержка и т. д.	Input
Правила и процедуры	Правила продаж, инструкции по сборке, процедуры тестирования, критерии производительности и т. д.	Control
Проданные продукты	Настольные и портативные компьютеры	Output
Бухгалтерская система	Оформление счетов, оплата счетов, работа с заказами	Mechanism

14. С помощью кнопки  внесите текст в поле диаграммы - точку зрения и цель (рисунок 7.11)

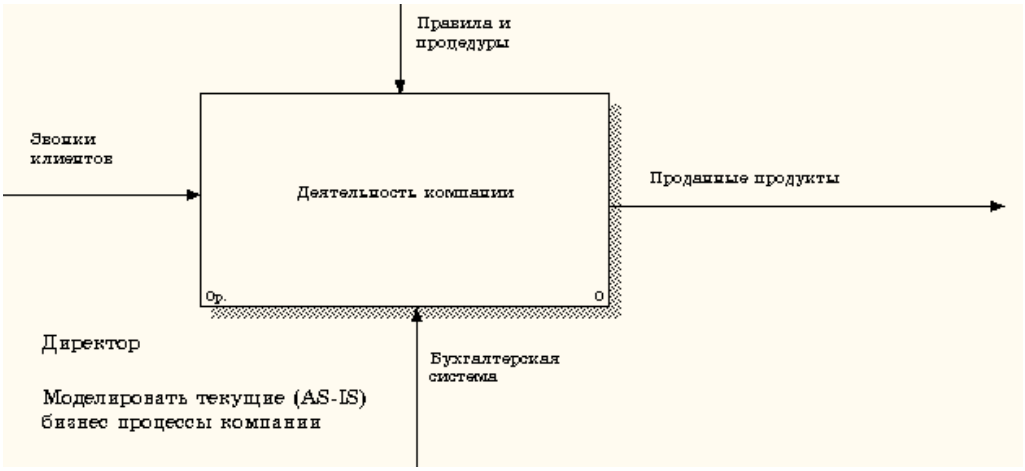


Рисунок 7.11 - Внесение текста в поле диаграммы с помощью редактора Text Block Editor

Результат выполнения показан на рисунке 7.12.

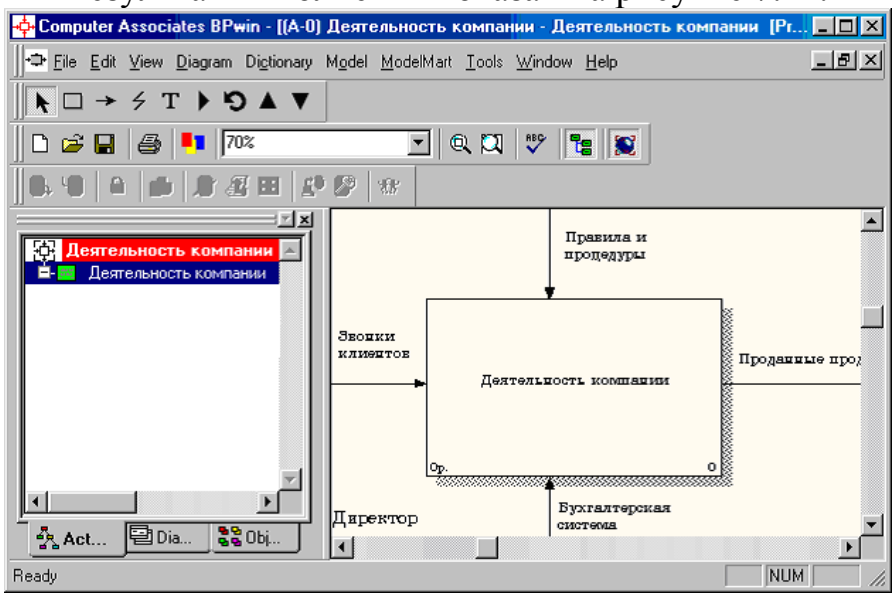


Рисунок 7.12 – Построенная контекстная диаграмма

15. Создайте отчет по модели. В меню **Tools/Reports/Model Report** (рисунок 7.13) задайте опции генерирования отчета (установите галочки) и нажмите кнопку **Preview** (Предварительный просмотр) (рисунок 7.14).

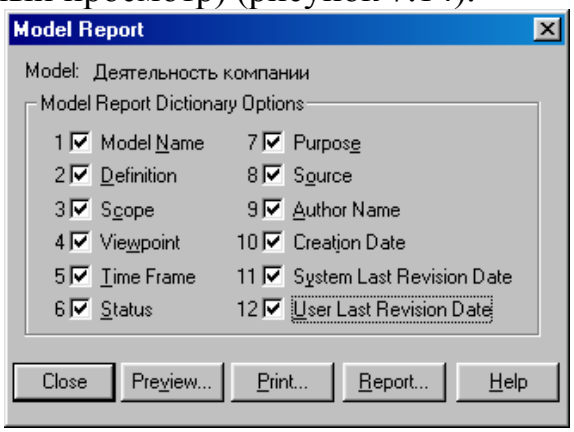


Рисунок 7.13 – Задание опций генерирования отчета **Model Report**

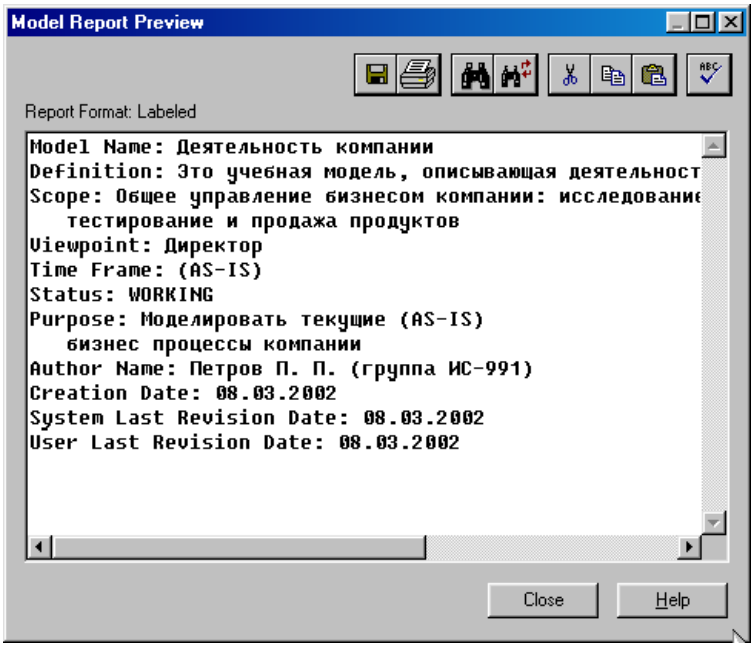


Рисунок 7.14 – Предварительный просмотр отчета **Model Report**

16. Выберите кнопку ▼ перехода на нижний уровень в палитре инструментов и в диалоговом окне **Activity Box Count** (рисунок 7.15) установите число работ на диаграмме нижнего уровня - 3 - и нажмите кнопку **ОК**.

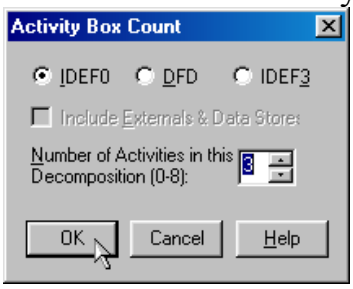


Рисунок 7.15 – Диалоговое окно **Activity Box Count**

17. Автоматически будет создана диаграмма декомпозиции (рисунок 7.16).

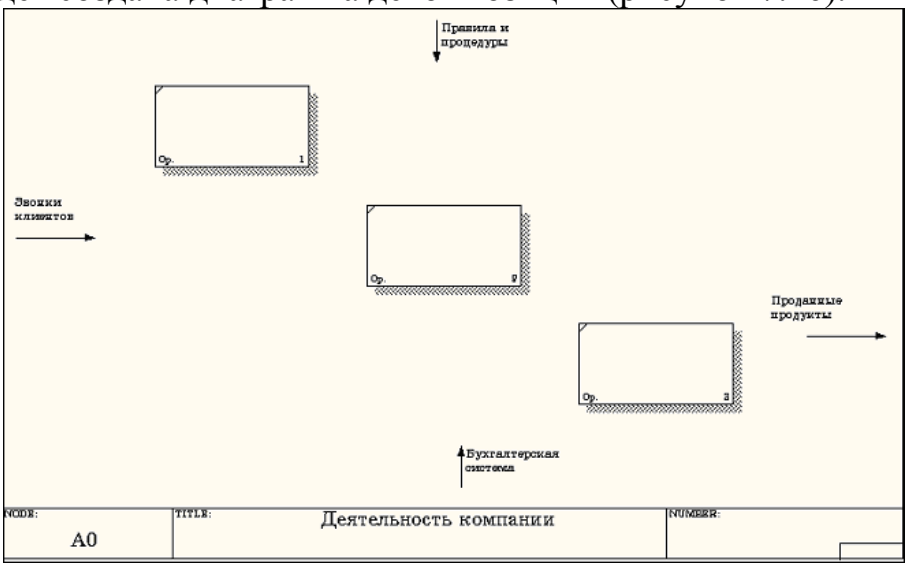


Рисунок 7.16 – Диаграмма декомпозиции

Правой кнопкой мыши щелкните по работе расположенной в левом верхнем углу области редактирования модели, выберите в контекстном меню опцию **Name** и внесите имя работы. Повторите операцию для оставшихся двух работ. Затем внесите определение, статус и источник для каждой работы согласно данным таблицы 7.2.

Таблица 7.2 - Работы диаграммы декомпозиции A0

Название работы (Activity Name)	**Определение работы** (Activity Definition)
Продажи и маркетинг	Телемаркетинг и презентации, выставки
Сборка и тестирование компьютеров	Сборка и тестирование настольных и портативных компьютеров
Отгрузка и получение	Отгрузка заказов клиентам и получение компонентов от поставщиков

Диаграмма декомпозиции примет вид представленный на рисунке 7.17.

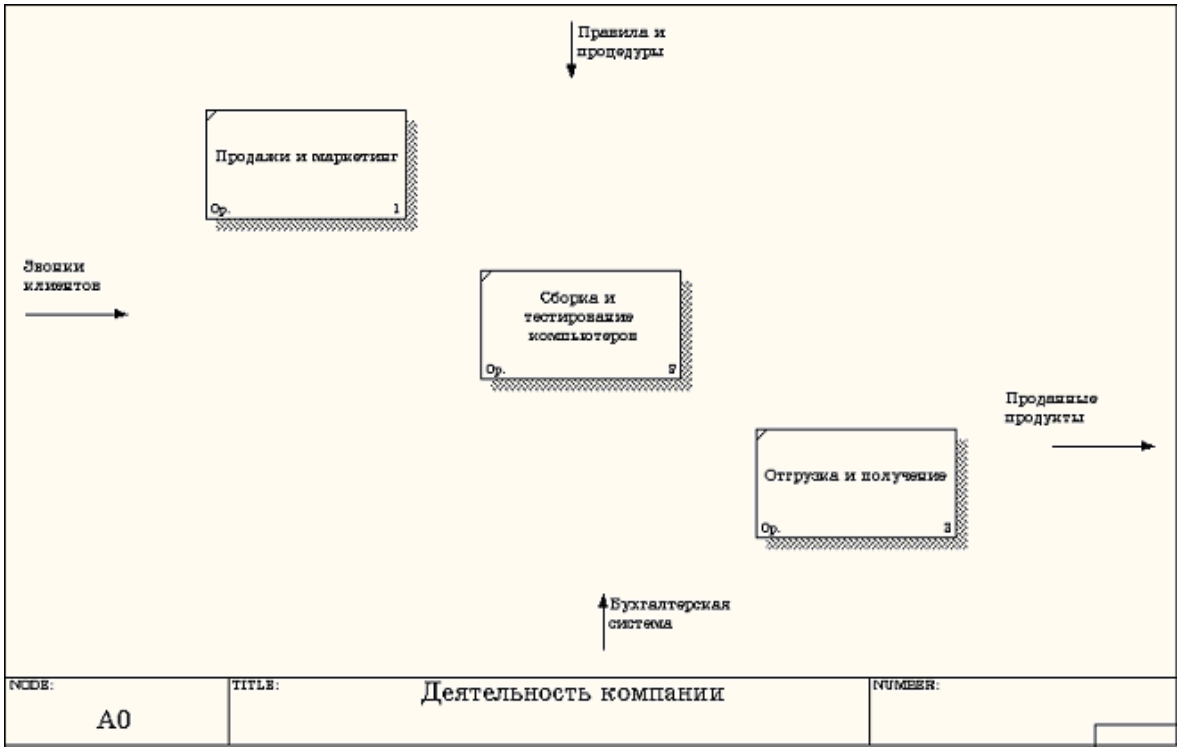


Рисунок 7.17 – Диаграмма декомпозиции после присвоения работам наименований

18. Для изменения свойств работ после их внесения в диаграмму можно воспользоваться словарем работ (рисунок 7.18). Вызов словаря производится при помощи пункта главного меню **Dictionary /Activity**.

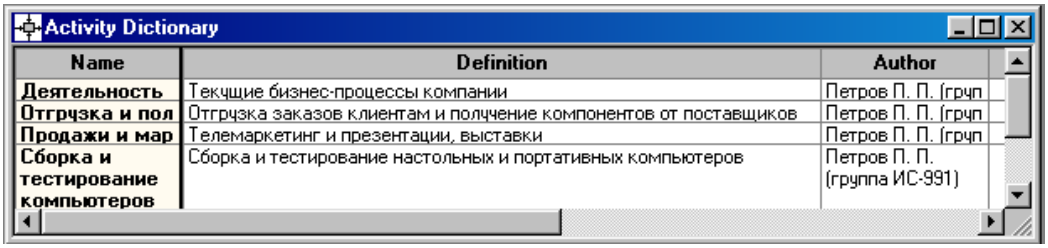


Рисунок 7.18 - Словарь Activity Dictionary

Если описать имя и свойства работы в словаре, ее можно будет внести в диаграмму позже с помощью кнопки в палитре инструментов. Невозможно удалить работу из словаря, если она используется на какой-либо диаграмме. Если работа удаляется из диаграммы, из словаря она не удаляется. Имя и описание такой работы может быть использовано в дальнейшем. Для добавления работы в словарь необходимо перейти в конец списка и щелкнуть правой кнопкой по последней строке. Возникает новая строка, в которой нужно внести имя и свойства работы. Для удаления всех имен работ, не использующихся в модели, щелкните по кнопке (**Purge (Чистить)**).

19. Перейдите в режим рисования стрелок и свяжите граничные стрелки, воспользовавшись кнопкой на палитре инструментов так, как это показано на рисунке 7.19.

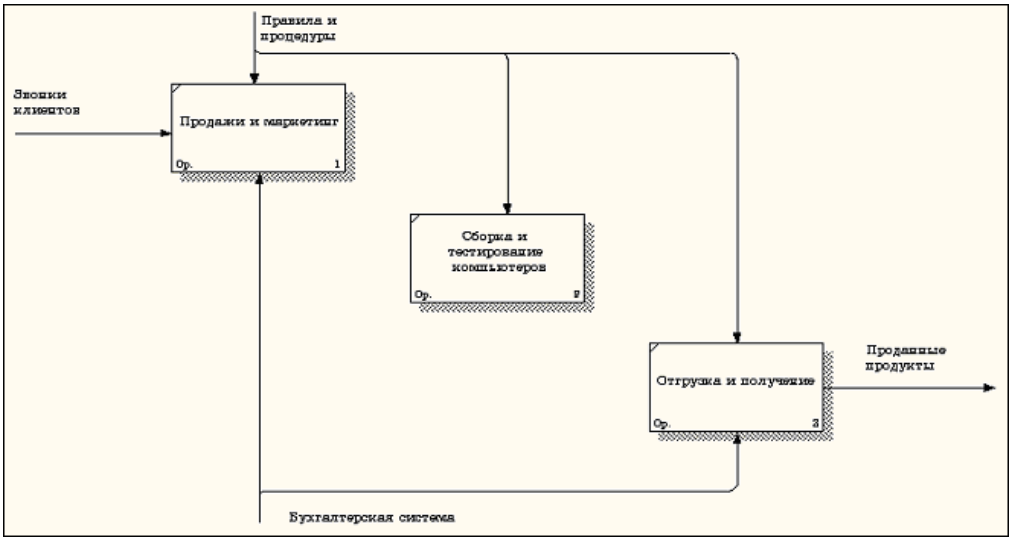


Рисунок 7.19 - Связанные граничные стрелки на диаграмме A0

20. Правой кнопкой мыши щелкните по ветви стрелки управления работы **"Сборка и тестирование компьютеров"**и переименуйте ее в **"Правила сборки и тестирования"**(рисунок 7.20).

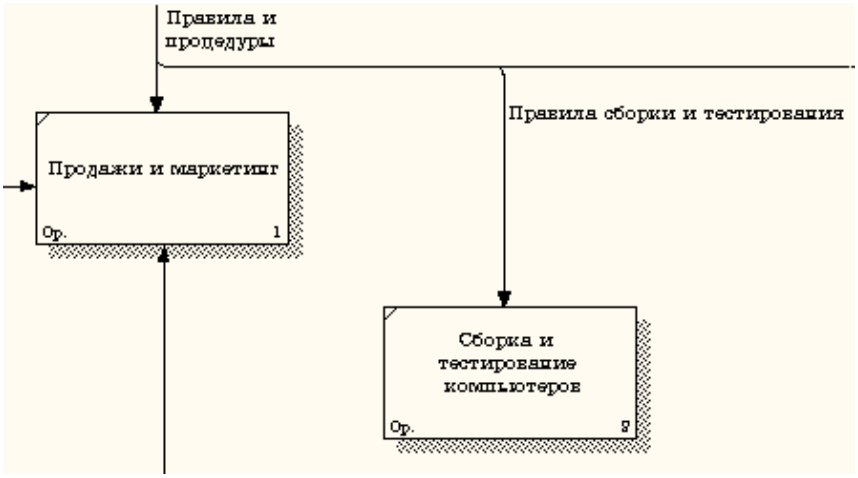


Рисунок 7.20 - Стрелка "Правила сборки и тестирования"

Внесите определение для новой ветви: "**Инструкции по сборке, процедуры тестирования, критерии производительности и т. д**." Правой кнопкой мыши щелкните по ветви стрелки механизма работы **"Продажи и маркетинг"**и переименуйте ее как **"Система оформления заказов" (**рисунок 7.21**)**.

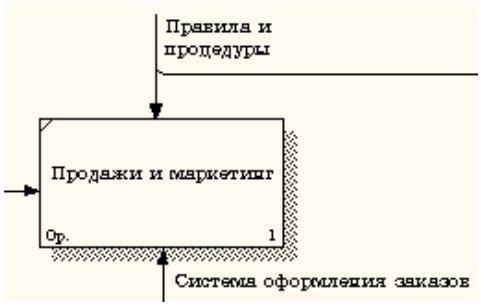
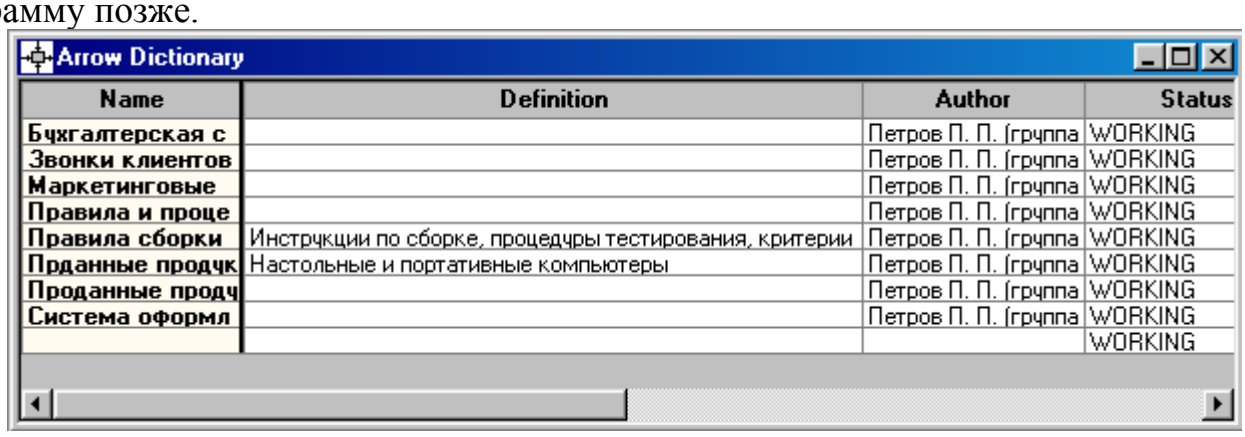


Рисунок 7.21 - Стрелка " Система оформления заказов "

21. Альт ернативный метод внесения имен и свойств стрелок - использование словаря стрелок (вызов словаря - меню **Dictionary/ Arrow**). Если внести имя и свойства стрелки в словарь (рисунок 7.22), ее можно будет внести в диаграмму позже.



Name	Definition	Author	Status
Бухгалтерская с		Петров П. П. (группа	WORKING
Звонки клиентов		Петров П. П. (группа	WORKING
Маркетинговые		Петров П. П. (группа	WORKING
Правила и проце		Петров П. П. (группа	WORKING
Правила сборки	Инструкции по сборке, процедуры тестирования, критерии	Петров П. П. (группа	WORKING
Проданные продук	Настольные и портативные компьютеры	Петров П. П. (группа	WORKING
Проданные проду		Петров П. П. (группа	WORKING
Система оформл		Петров П. П. (группа	WORKING
			WORKING

Рисунок 7.22 – Словарь стрелок

Стрелку нельзя удалить из словаря, если она используется на какой-либо диаграмме. Если удалить стрелку из диаграммы, из словаря она не удаляется. Имя и описание такой стрелки может быть использовано в дальнейшем. Для добавления стрелки необходимо перейти в конец списка и

щелкнуть правой кнопкой по последней строке. Возникает новая строка, в которой нужно внести имя и свойства стрелки.

22. Создайте новые внутренние стрелки так, как показано на рисунке 7.23.

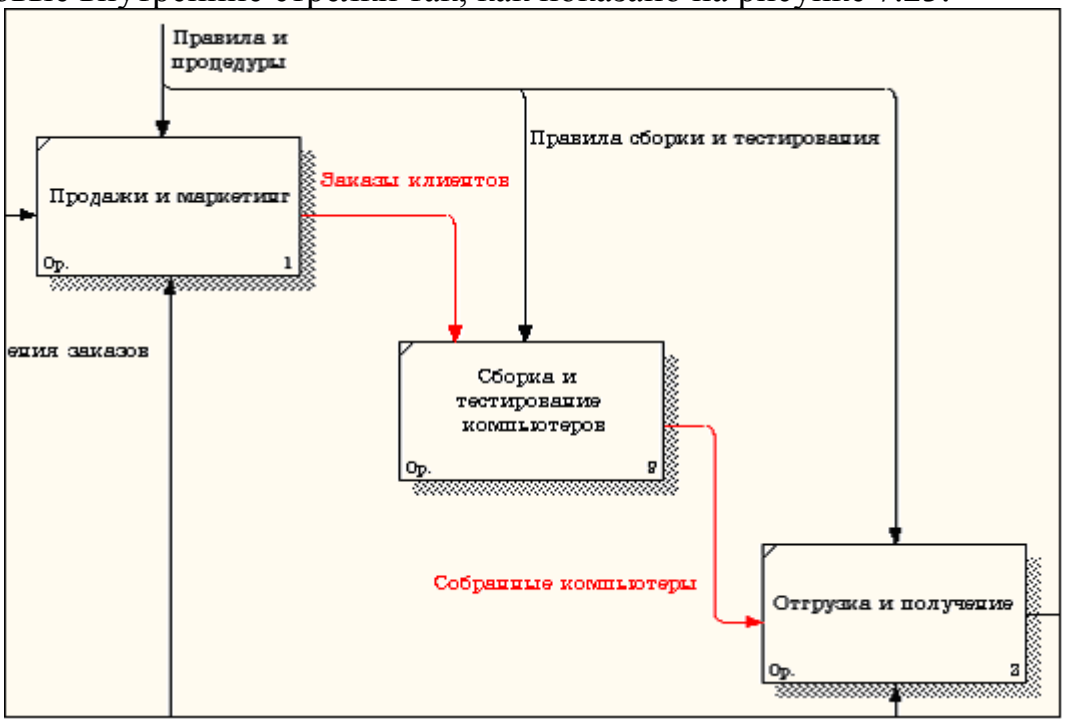


Рисунок 7.23 - Внутренние стрелки диаграммы A0

23. Создайте стрелку обратной связи (по управлению) **"Результаты сборки и тестирования",** идущую от работы **"Сборка и тестирование компьютеров"**к работе **"Продажи и маркетинг"**.Измените, при необходимости, стиль стрелки (толщина линий) и установите опцию **Extra Arrowhead (**Дополнительный Наконечник стрелы**)** (из контекстного меню). Методом **drag&drop** перенесите имена стрелок так, чтобы их было удобнее читать. Если необходимо, установите из контекстного меню **Squiggle** (Загогулину). Результат возможных изменений показан на рисунке 7.24.

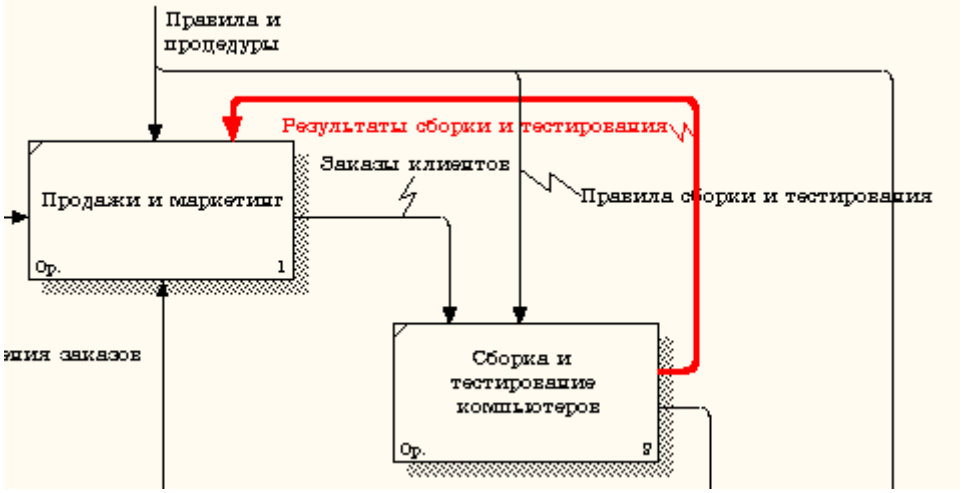


Рисунок 7.24 - Результат редактирования стрелок на диаграмме A0

24. Создайте новую граничную стрелку выхода **"Маркетинговые материалы",**

выходящую из работы **"Продажи и маркетинг".**Эта стрелка автоматически не попадает на диаграмму верхнего уровня и имеет квадратные скобки на наконечнике (рисунок 7.25).

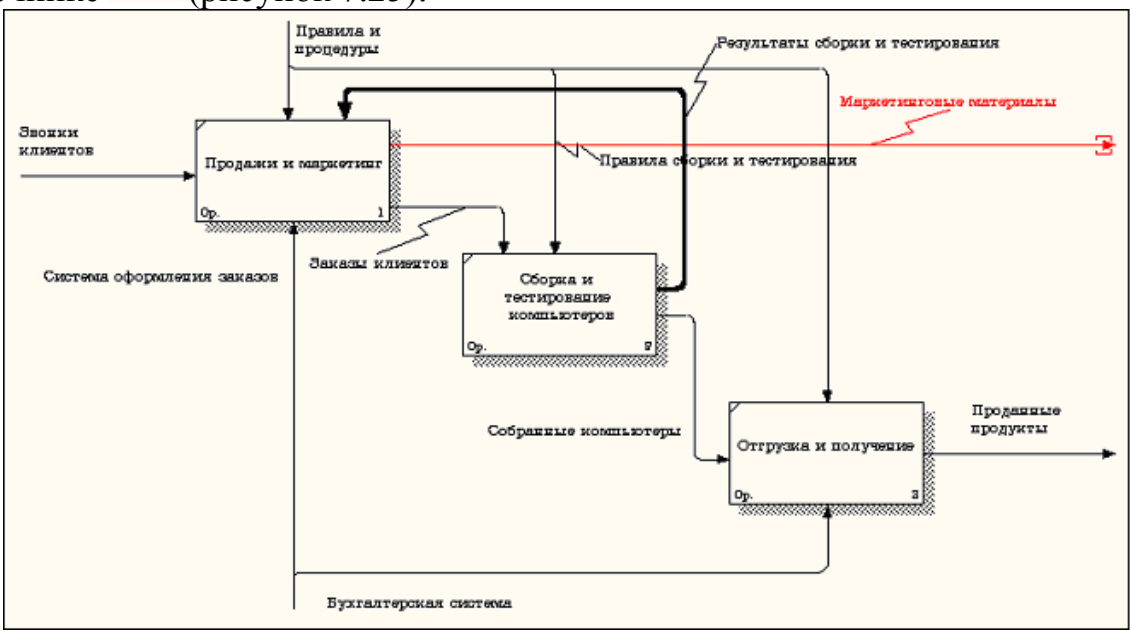


Рисунок 7.25 – Стрелка **Маркетинговые материалы**

25. Щелкните правой кнопкой мыши по квадратным скобкам и выберите пункт меню **Arrow Tunnel** (рисунок 7.26).

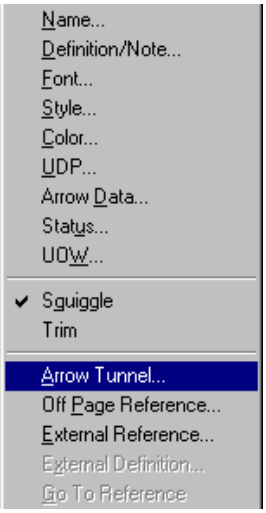


Рисунок 7.26 - Пункт меню **Arrow Tunnel**

В диалоговом окне **Border Arrow Editor (**Редактор Граничных Стрелок**)** выберите опцию **Resolve it to Border Arrow** (Разрешить как Граничную Стрелку) (рисунок 7.27).

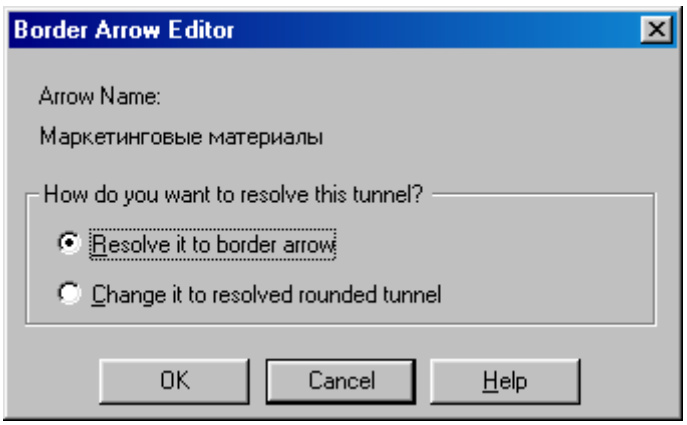


Рисунок 7.27 – Диалоговое окно **Border Arrow Editor**

Для стрелки **"Маркетинговые материалы"**выберите опцию **Trim (**Упорядочить**)** из контекстного меню. Результат выполнения работы показан на рис. 7.28

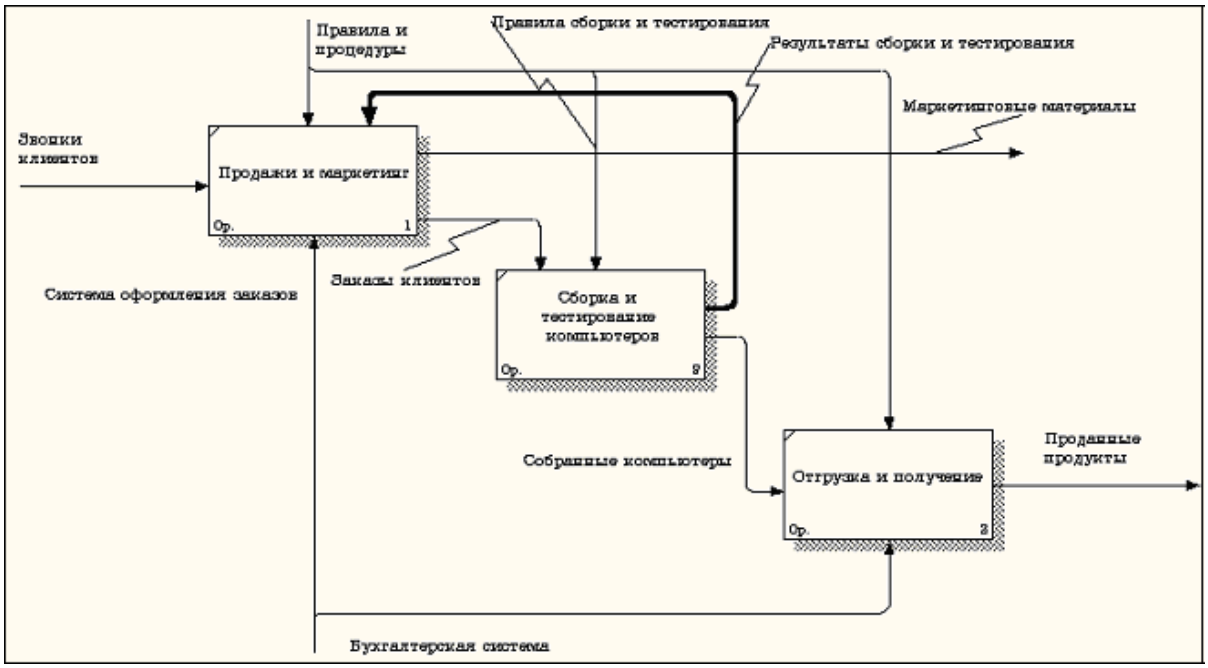


Рисунок 7.28 - Результат выполнения работы

8. Контрольные вопросы

1. Какие методологии поддерживает CASE-| средство BPwin|?
2. Назовите цель построению диаграмм с применением IDEF0.
3. Что представляет собой модель с применением IDEF0?
4. Перечислите виды диаграмм, которые входят в модель IDEF0.
5. Как располагаются работы на диаграммах декомпозиции?
6. Перечислите виды стрелок, применяющихся в диаграммах.
7. Как выполняется|исполняет| декомпозиция диаграмм?
8. Для чего предназначен словарь стрелок?
9. Что показывает диаграмма дерева узлов?
10. Какие отчеты можно создать по модели IDEF0?
11. Перечислите составные элементы диаграммы IDEF0.