Литература

1. К. Ларман

Применение UML 2.0 и шаблонов проектирования.

Введение в объектно-ориентированный анализ, проектирование и итеративную разработку.

Третье издание, Вильямс, 2013 год.

2. Г. Буч

UML.

Второе издание, Питер, 2006 год.

3. Л. Мацяшек

Анализ и проектирование информационных систем с помощью UML 2.0

Третье издание, Вильямс, 2008 год.

4. Г. Буч, А. Якобсон, Дж. Рамбо

UML. Классика CS.

Второе издание, Питер, 2006 год.

5. А.Леоненков

Объектно-ориентированный анализ и проектирование с использованием UML и Rational Rose.

Бином, 2006 год.

Литература (продолжение)

6. Р. Фатрелл, Д. Шафер, Л. Шафер
Управление программными проектами.
Вильямс, 2006 год.
8. Г. Буч и др.
Объектно-ориентированный анализ и проектирование с примерами приложений
Третье издание, Вильямс, 2008 год.
9. Д. Карлсон
Eclipse
Из-во «Лори», 2013
11. Сайт университета информационных технологий.
www.intuit.ru

Информационные технологии и ПО

- ИТ- технологии хранения, обработки и передачи информации, ориентированные на использование компьютеров.
- ПО – совокупность программ, связанных с ними данных и документации.
- Основная мировая тенденция – темпы роста затрат на ИТ, включая ПО, превышают темпы роста экономики в целом.

Стандарты документирования ПС

- Стандарты Institute of Electrical and Electronics Engineers

IEEE 830-1993, IEEE 1016-1998

- Государственные стандарты СССР

ГОСТ 34.602-89, 34.201-89

- Руководящие документы по стандартизации

РД 50-34.698-90 (действует с 1992 года)

- Государственные стандарты России

ГОСТ Р ИСО/МЭК 12207-2010

Обобщенные критерии качества

1. Функциональность Functionality
2. Мобильность Mobility
3. Надежность Reliability
4. Эффективность Performance
5. Сопровождаемость Serviceability

(модифицируемость)

6. Практичность Usability

(понятность и простота использования)

Стратегии разработки ПО

Функционально-ориентированная стратегия

технологии:

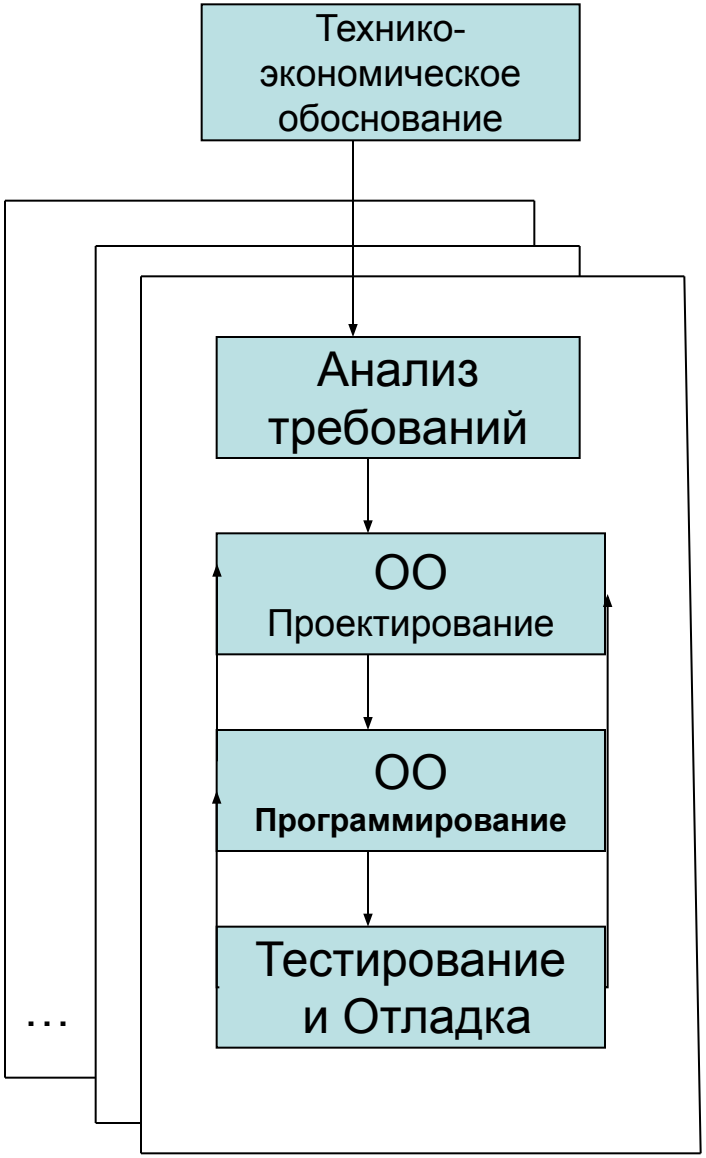
- Нисходящая (водопадная)
- Расширения ядра

Объектно-ориентированная стратегия

технологии:

- Спиральная
- Эволюционная или инкрементная
- Гибкая (agile software development)

ОО-технологии: этапы разработки



Эволюция начинается
со второй итерации

Достоинства ОО-технологий разработки ПО

1. Тесная связь с заказчиком в процессе разработки.
2. Возможность изменения требований к ПО.
3. Получение работающих версий до завершения разработки.
4. Повышенное внимание к объектам и структурам данных.
5. Возможность принятия альтернативных решений.
6. Детальная отработка элементов интерфейса.
7. Равномерное распределение разных видов работ в процессе создания программной системы.

Для чего нужны технологии?

1. Повысить качество ПО.
2. Снизить сроки разработки.
3. Уменьшить стоимость.
4. Обеспечить контроль над ходом разработки.
5. Уменьшить риски.

CASE-средства

CASE (Computer Aided Software Engineering)-средства ориентированы на постоянное использование компьютера в процессе разработки ПО.

В большинстве CASE-средств применяются UML-диаграммы.

Наиболее известные CASE-средства – Rational Software Architect (IBM), Together (Borland), AllFusion (Computer Associates)

Поддержка UML-диаграмм встроена во многие системы программирования: Visual Studio, Delphi.

http://www.objectsbydesign.com/tools/umltools_byCompany.html

Цели использования CASE-средств:

- построение UML-диаграмм;
- генерация кода по UML-диаграммам;
- генерация UML-диаграмм на основе кода.

Интерфейс: удобство и эстетичность

Факторы:

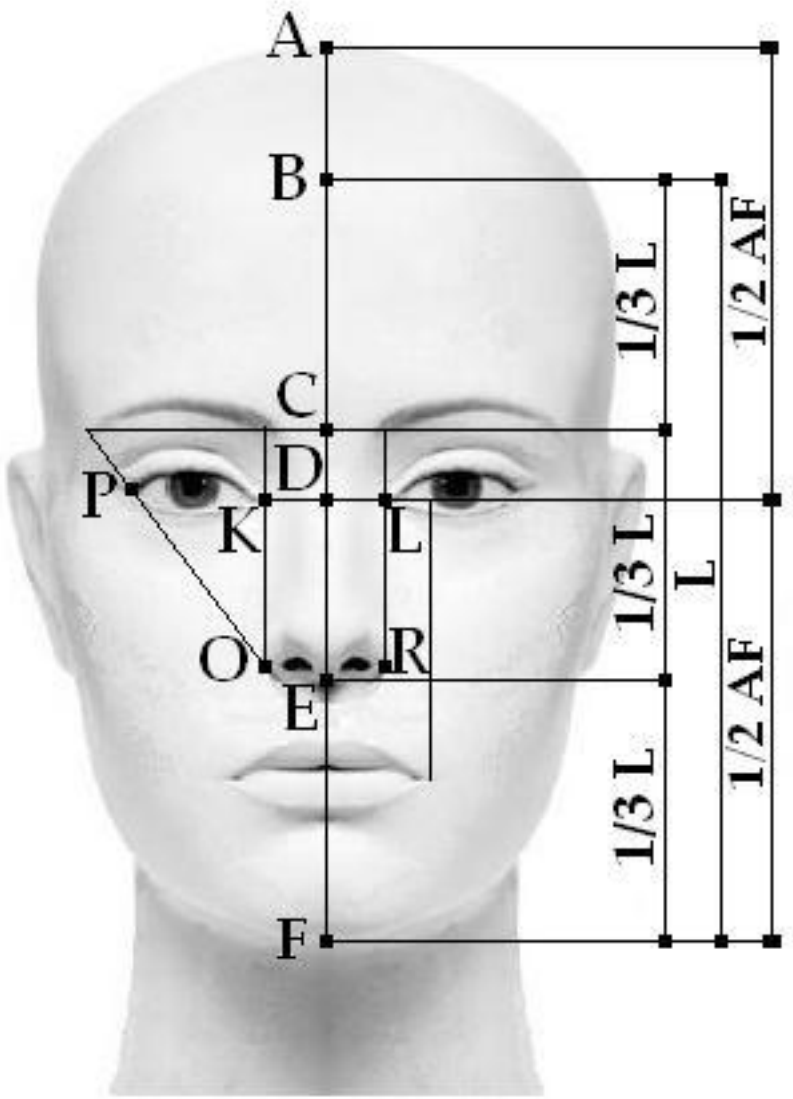
- Цветовая гамма
- Симметрия
- Выравнивание
- Расстояния между элементами
- Пропорциональность
- Группирование связанных элементов
- Порядок расположения

Красота

BC=CE=EF

AD=DF

OR=KL=PK



Этап программирования

Может быть разбит на два следующих друг за другом подэтапа:

- разработка алгоритма решения;
- кодирование или написание кода программы.

Для представления разрабатываемого алгоритма могут использоваться:

- структурные схемы,
- диаграммы деятельности,
- псевдокод.

Структурные схемы

Структурная схема программы – это ориентированный граф, задающий порядок выполнения операторов; вершинам графа соответствуют операторы, а ребра определяют последовательность их выполнения.

Простые и непростые структурные схемы. Структурная схема является простой, если она:

- имеет один вход;
- имеет один выход;
- через каждый ее элемент проходит по крайней мере один путь от входа к выходу.

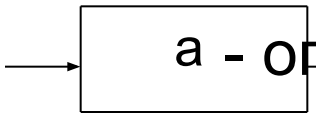
Основная теорема

Структурная схема любого алгоритма обработки данных может быть представлен в виде суперпозиции произвольного числа трех базовых управляющих структур: следования, выбора и повторения.

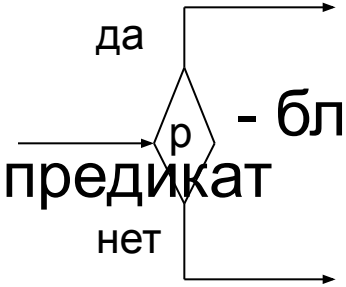
Структурные схемы программ

Условные обозначения:

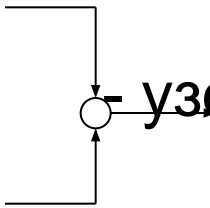
- линия передачи управления



- операционный блок, «a» - функция



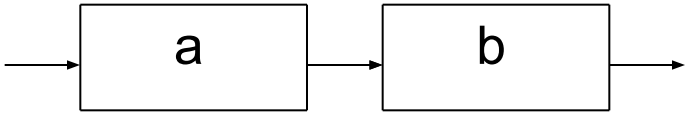
- блок принятия решения, «p» - предикат



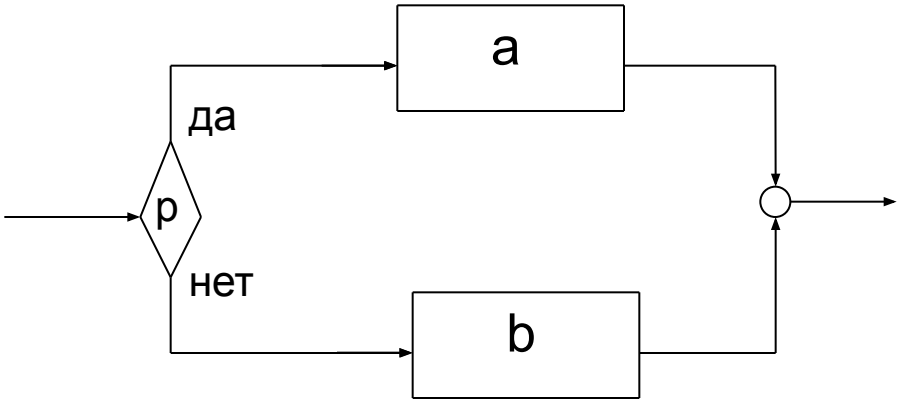
- узел слияния

Базовые управляющие структуры

1. Следование

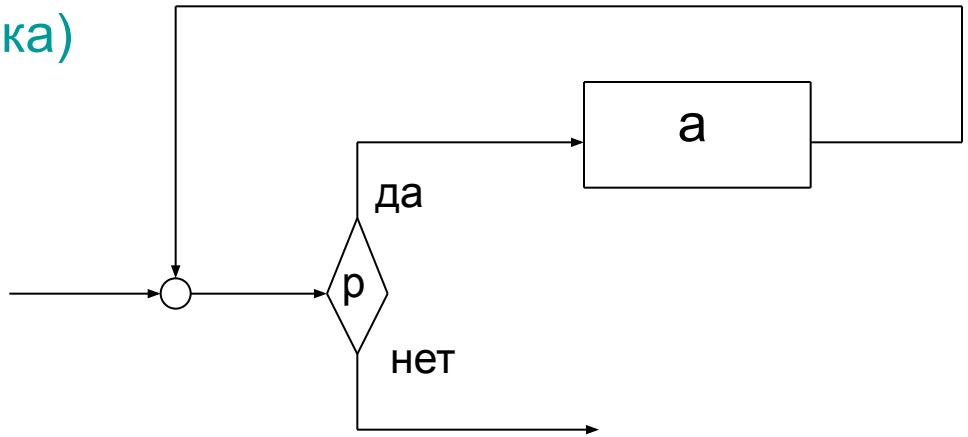


2. Выбор

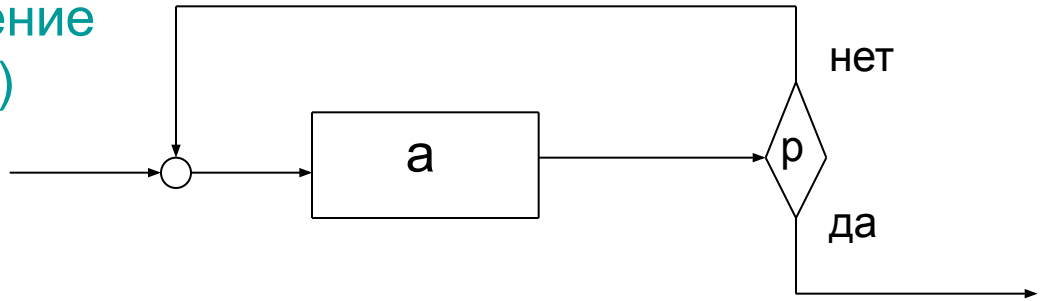


Базовые управляющие структуры (продолжение)

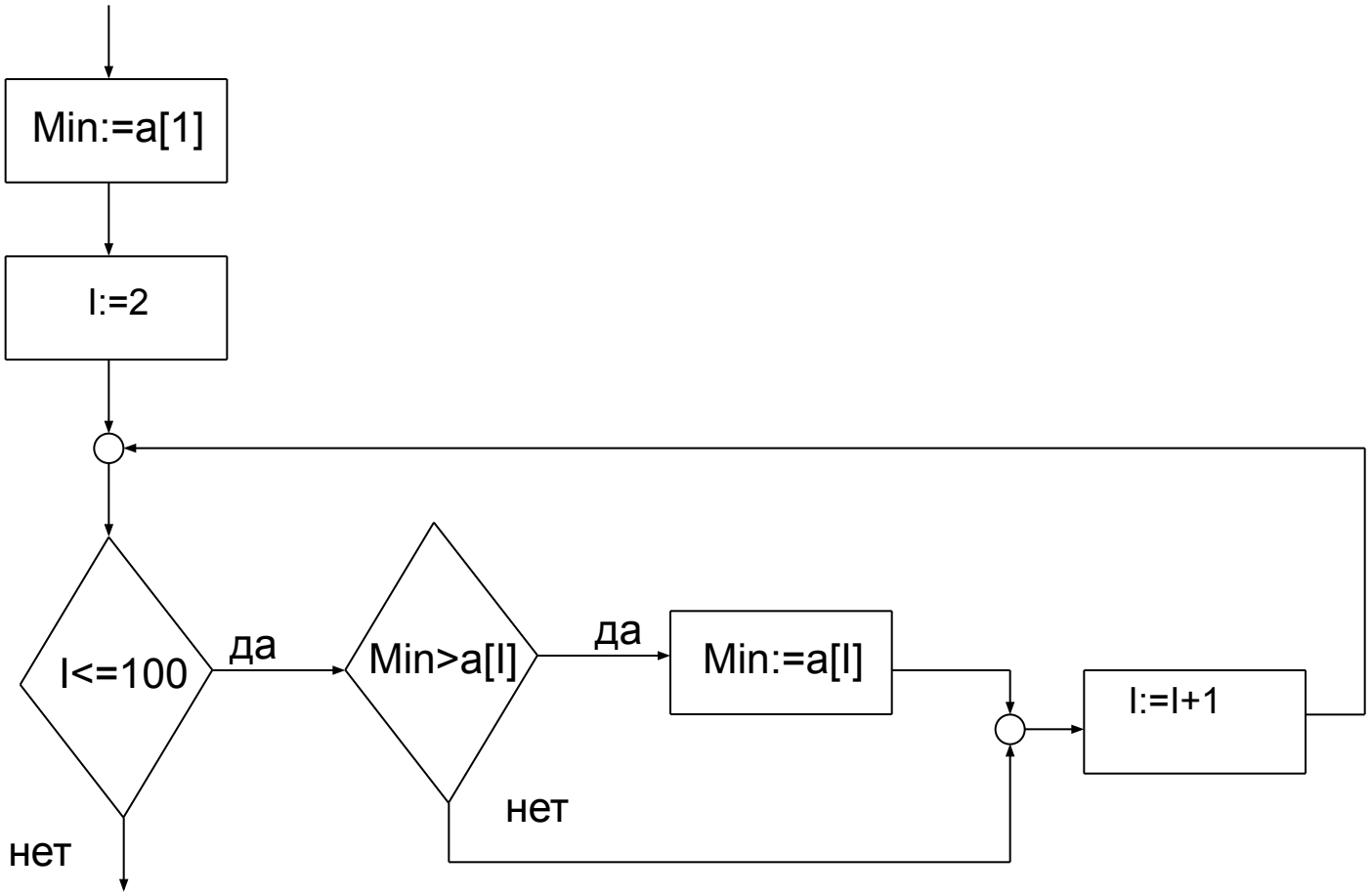
3 а. Повторение (Цикл-пока)



3 б. Повторение (Цикл-до)

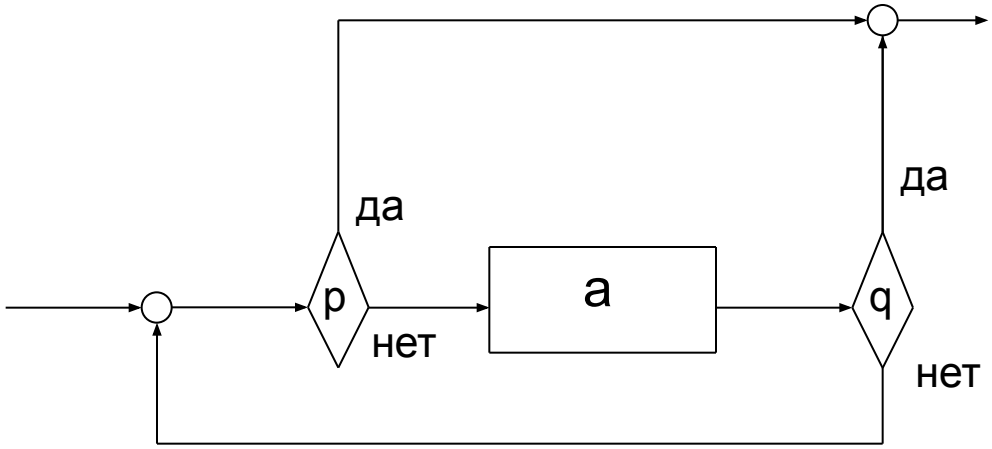


Декомпозиция структурных схем



Теорема о декомпозиции

Структурная схема разложима на базовые управляющие структуры тогда и только тогда, когда она не содержит замкнутого контура с более чем одним выходом из него.



Теорема о соотношениях элементов структурных схем

Предположим, что в простой структурной схеме имеется:

f – операционных блоков;

p – блоков принятия решения;

g – узлов слияния;

t – линий передачи управления.

Тогда справедливо:

(1) $p = g$

(2) $t = f + 3p + 1$

	Вход	Операц. блоки	Блоки прин. реш.	Узлы слияния	Выход
Начальные точки	1	f	2p	g	-
Конечные точки	-	f	p	2g	1

Семинар №1

1. Привести примеры непростых структурных схем.
2. Выразить Цикл-Пока через остальные базовые управляющие структуры.
3. Выразить Цикл-До через остальные базовые управляющие структуры.
4. Нарисовать схему алгоритмически эквивалентную заданной и состоящую только из базовых управляющих структур.

Диаграммы деятельности

Также как структурные схемы, служат для описания алгоритмов.

Диаграммы деятельности содержат следующие элементы:

1. Деятельности – действия.
2. Линии передачи управления.
3. Синхронизационные линии.
   - join - объединение
   - fork – распараллеливание
4. Блоки принятия решений.
5. Узлы слияния.
6. Вовлеченные объекты.
7. Символы развертывания.

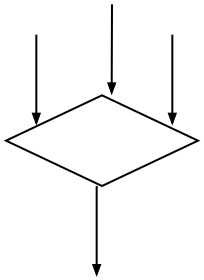
Диаграммы деятельности (обозначения)

Начало

Конец

Линия передачи управления

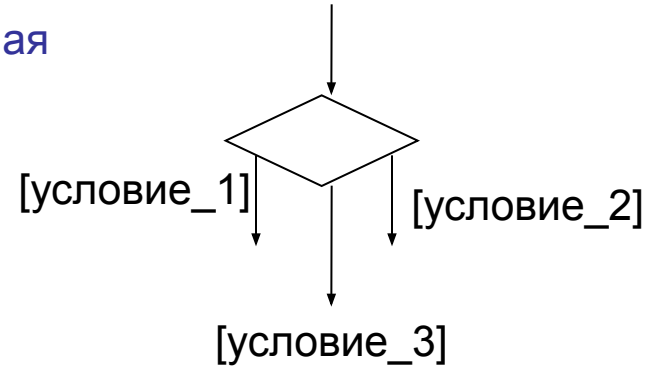
Синхронизационная черта



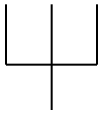
Узел слияния

Деятельность

Вовлеченный объект

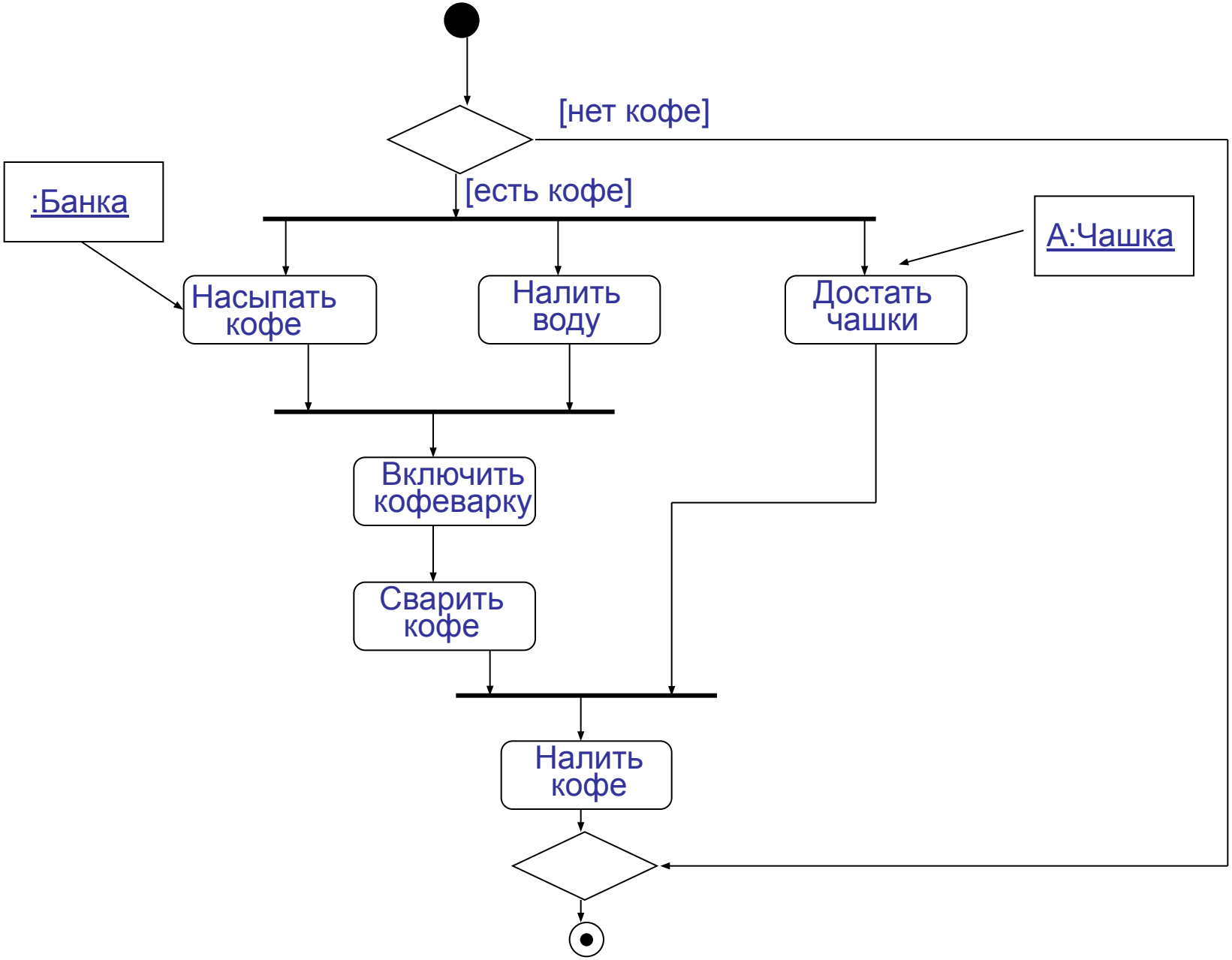


Блок принятия решения



Символ развертывания

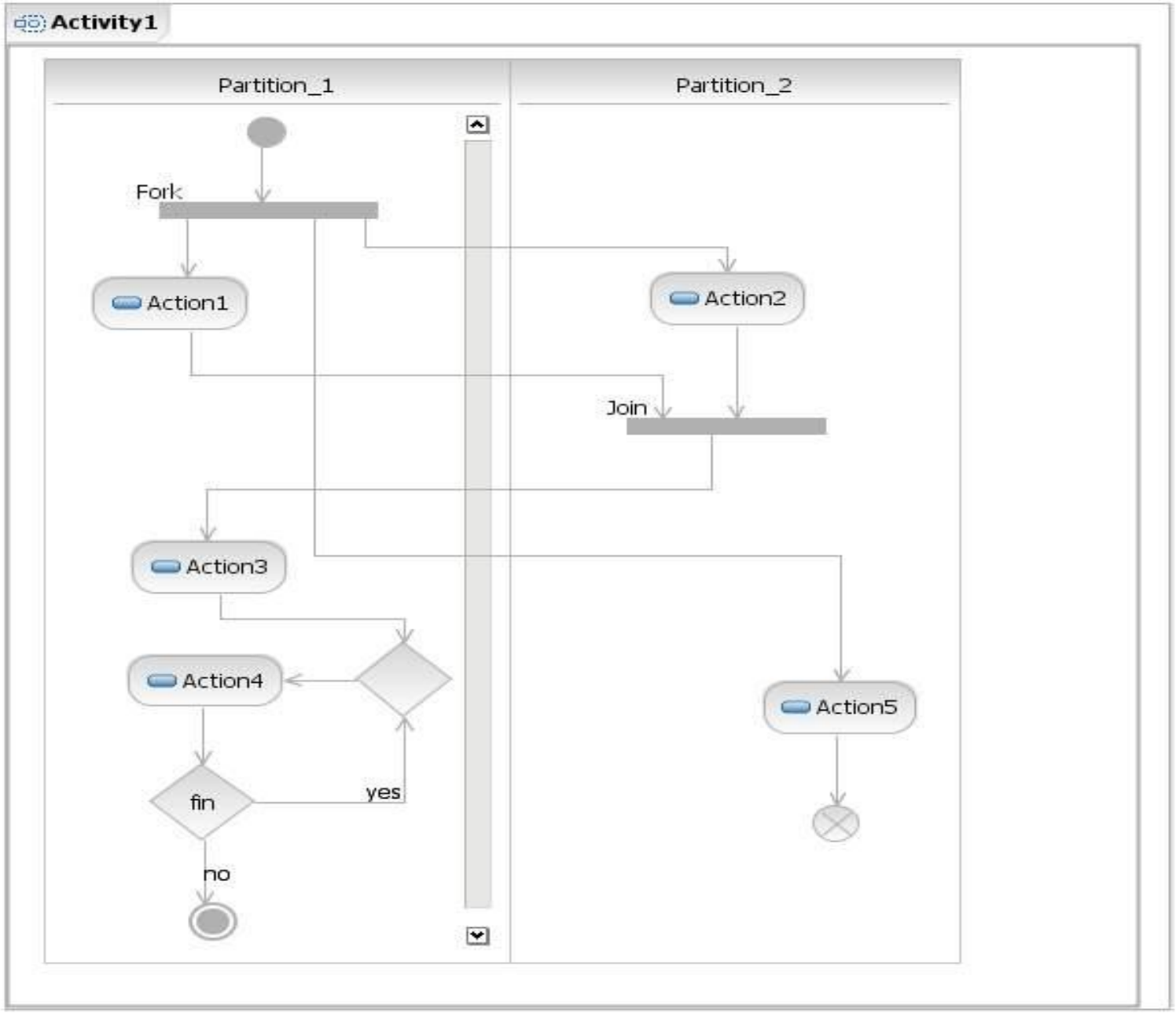
Диаграммы деятельности



Принципы построения диаграмм деятельности

1. Представлять только те детали, которые соответствуют данному уровню абстракции.
2. Основное внимание уделять главному потоку управления.
3. Число пересечений линий передачи управления должно быть минимальным.
4. Обязательно изображать вовлеченные объекты.
5. Следить за использованием синхронизационных линий.
6. Как и в псевдокоде отображать не вполне определенные действия, имея в виду их дальнейшую детализацию.

Диаграмма деятельности с дорожками



Псевдокод

1. Каждое предложение-действие записывается на отдельной строке.

2. Конструкция ЕСЛИ-ТО-ИНАЧЕ

```
ЕСЛИ <условие> ТО
    <действие_1>
ИНАЧЕ
    <действие_2>
ВСЕ-ЕСЛИ
```

3. Конструкция ЦИКЛ-ПОКА

```
ЦИКЛ-ПОКА <условие>
     <действие>
ВСЕ-ЦИКЛ
```

Псевдокод (продолжение)

4. Конструкция ЦИКЛ-ДО

```
ЦИКЛ-ДО
    <действие>
ВСЕ-ЦИКЛ <условие>
```

5. Конструкция ВЫБОР

```
ВЫБОР <переключатель>
    ключ_1: <действие_1>
      …
    ключ_n: <действие_n>
ВСЕ-ВЫБОР
```

6. Конструкция ИСКЛЮЧИТЕЛЬНАЯ СИТУАЦИЯ

```
СИТУАЦИЯ <описание>
    <реакция>
ВСЕ-СИТУАЦИЯ
```

Пошаговая детализация

Пошаговая детализация – это процесс, который используется для декомпозиции функции каждого метода или подпрограммы в соответствии с логикой его работы. Процесс заключается в разбиении функции на подфункции. В конце концов все сводится к описанию шагов конкретного алгоритма.

Все функции и подфункции записываются на псевдокоде. Пошаговая детализация – итерационный процесс, в котором на каждом шаге рассматриваются все более подробные детали и логика работы описывается все точнее. Первоначальный вариант – общая спецификация метода или подпрограммы.

Семинар №2

1. Написать на псевдокоде алгоритм определения квадратов целых чисел в диапазоне от 1 до 1000, одинаково читающихся в обоих направлениях (палиндромов).

   Madam, I’m Adam

2. Построить диаграмму деятельности для процесса сдачи экзамена студентом преподавателю.

Палиндромы

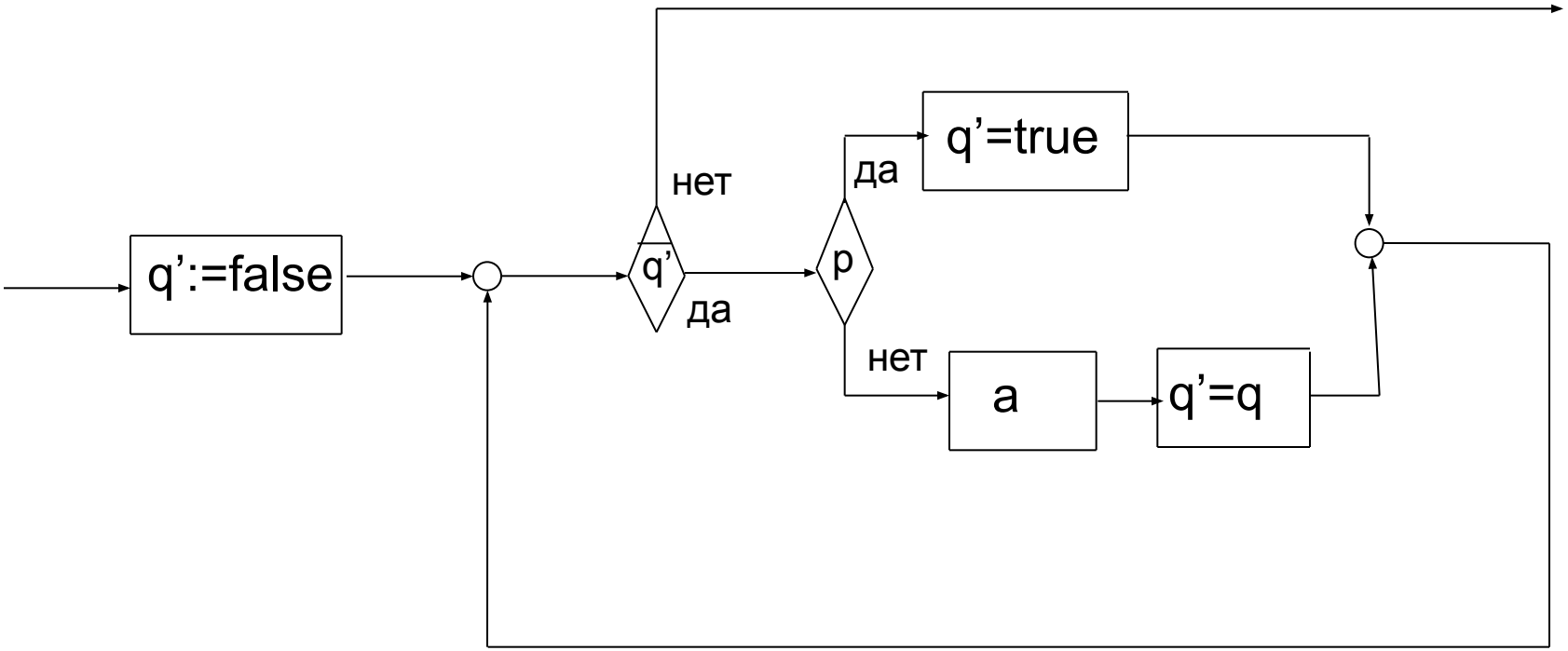
Число Лишрел— это натуральное число, которое не может стать палиндромом с помощью итеративного процесса «перевернуть и сложить» в десятичной системе счисления.

Например, берем число 57:

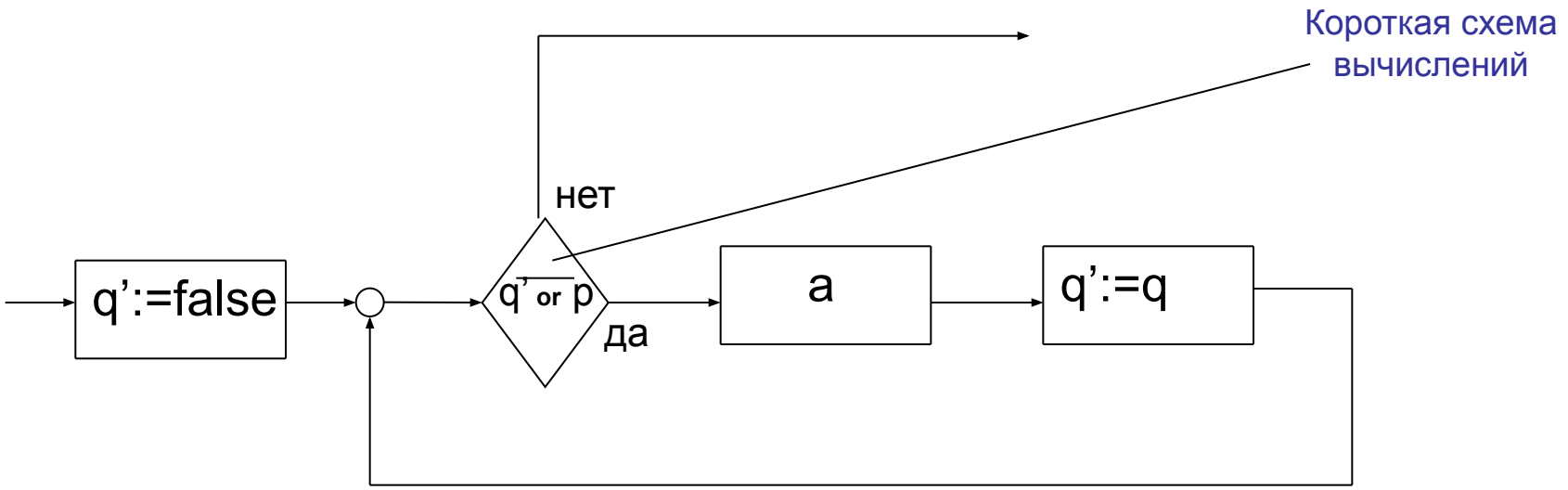
57 + 75 = 132 132 + 231 = 363

Проверено число 196 до 600 млн цифр и палиндром не обнаружен.

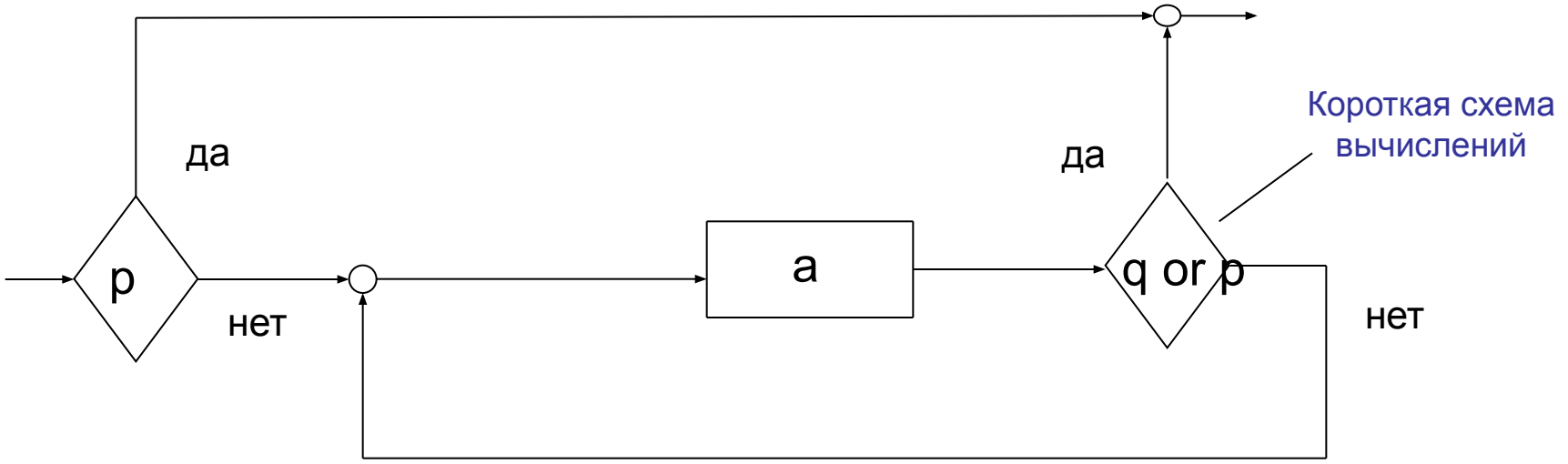
Структурная схема, состоящая из базовых управляющих конструкций



Структурная схема, состоящая из базовых управляющих конструкций



Структурная схема, состоящая из базовых управляющих конструкций



Рефакторинг

Изменение кода программы с целью улучшения ее качества без изменения реализуемых программой функций.

Оптимизация циклов

1.Вынесение операторов за пределы цикла

```
For I:=1 to 100 do
  Begin
     J:=2*K+I;
     A[J]:=K-1;
  End;
```

Оптимизация циклов

1.Вынесение операторов за пределы цикла

```
For I:=1 to 100 do
  Begin
     J:=2*K+I;
     A[J]:=K-1;
  End;
```

```
N:=K+K;
M:=K-1;
For I:=1 to 100 do
   A[N+I]:=M;
```

Оптимизация циклов (продолжение)

2. Развертывание цикла

```
For I:=1 to 1000 do
   A[I]:=1;
```

```
For I:=1 to 500 do
   Begin
      A[I]:=1;
      A[I+500]:=1;
   End;
```

3. Переупорядочивание циклов

```
For I:=1 to 20 do
   For J:=1 to 10 do
      A[I,J]:=1;
```

```
For J:=1 to 10 do
   For I:=1 to 20 do
      A[I,J]:=1;
```

Оптимизация логических выражений

1. Логическое ИЛИ

If (P or Q) then… If (Q or P) then…

2. Логическое И

If (P and Q) then… If (Q and P) then…

Оптимизация возможна только в том случае, когда используется короткая схема вычислений

Экономия оперативной памяти

- Использование динамической памяти
- Упаковка данных
- Передача аргументов по адресу, а не по значению
- Применение альтернативных структур хранения

Способы организации данных на внешних носителях

- Последовательная организация
- Прямая организация
- Индексно-последовательная организация

Данные размещаются в файлах, файлы состоят из записей, а запись представляет собой последовательность полей.

.

Пример записи

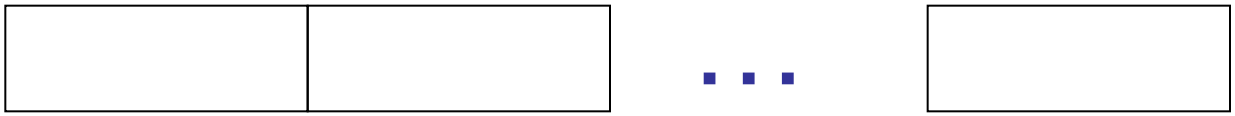
event_date	rq_uid	event_msg	proc_status	event_source	event_receiver	operation	source_queue	source_qm	target_queue	target_qm
9.11.14 17:27:24,321000000 +03:00	C70726369FE111e48B0800144FFA005C	Тело сообщения	SUCCEES	A	ESB	Получен запрос от A	A.IN	M1_A	ROUTER.IN	M1_ESB
9.11.14 17:27:24,324000000 +03:00	C70726369FE111e48B0800144FFA005C	Тело сообщения	SUCCEES	ROUTER	ESB	Сообщение успешно смаршрутизировано	ROUTER.IN	M1_ESB	B.IN	M1_B

Последовательная организация

В случае использования последовательной организации записи располагаются в памяти последовательно одна за другой. И доступ к ним возможен только последовательный.

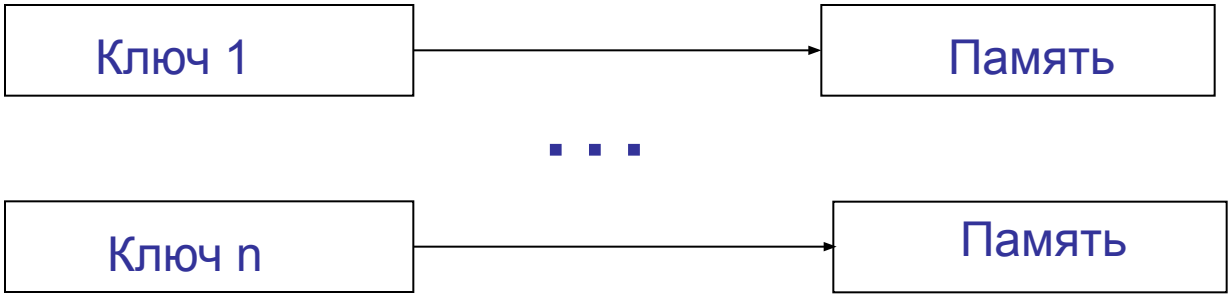
Никакой дополнительной информации о порядке размещении записей нет.

Для ленточных носителей – это единственно возможная организация.

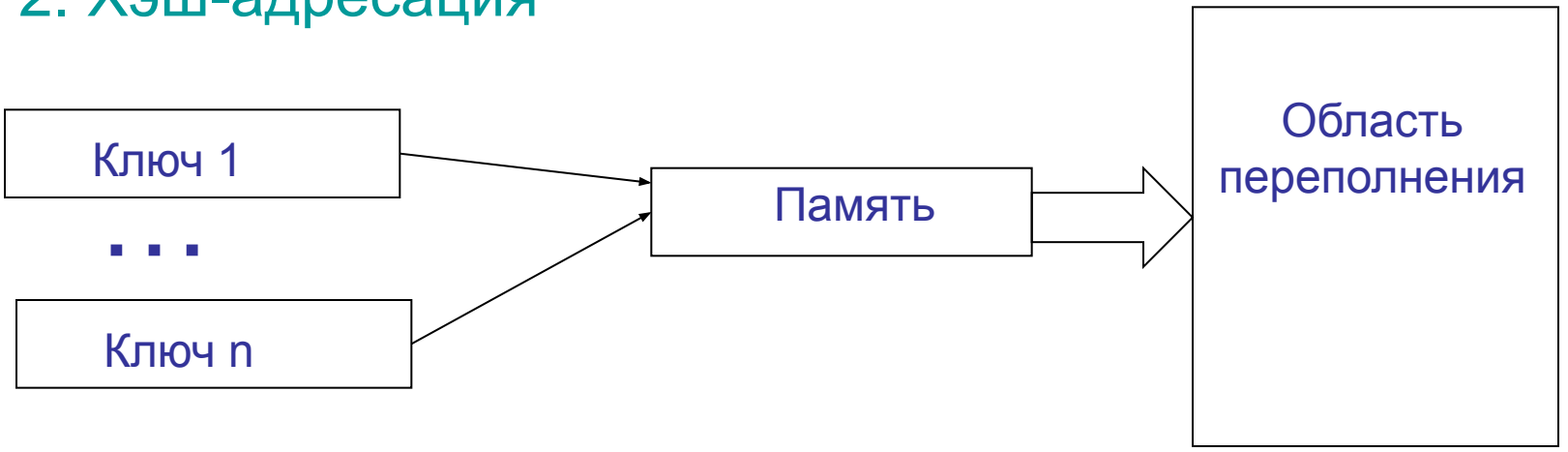


Прямая организация

1. Прямая адресация



2. Хэш-адресация



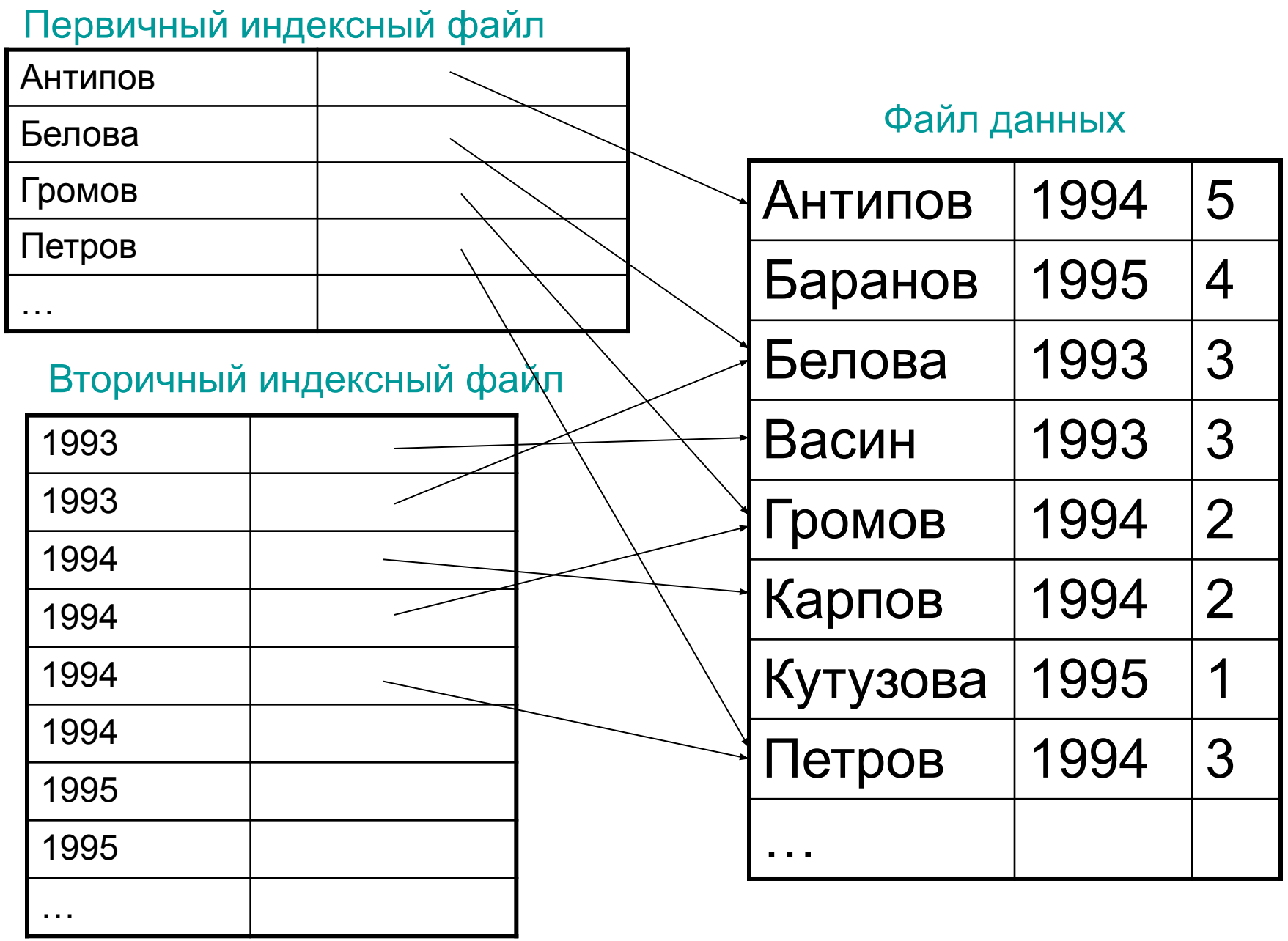
Пример хэш-функции

Значения первичного ключа K лежат в диапазоне 00..99, размер записи равен 50 байтов.

Известно, что возможные значения ключа распределены равномерно в указанном диапазоне и что число реальных записей не должно превышать 20.

Определить хэш-функцию для файла, состоящего из таких записей. Считать, что адресация памяти начинается с нуля.

Индексно-последовательная организация



Диаграммы развертывания

Диаграммы развертывания моделируют архитектуру систем во время работы (run-time)

Служат для описания конфигурации аппаратных средств и показывает, как элементы ПО (артефакты) связаны с аппаратными средствами.

Узел – это или элемент оборудования, который чаще всего обладает вычислительной мощностью и памятью, или ПО. Изображается в виде куба.

Узлы бывают двух типов. Устройство (*device*) – это физическое оборудование: компьютер или устройство, связанное с системой.

Среда выполнения (*execution environment*) – это программное обеспечение. Например, операционная система, система управления БД, виртуальная машина и т.п.

Диаграммы развертывания

Артефакт – это элемент, который является физическим воплощением программного обеспечения; обычно это файл. Такими файлами могут быть исполняемые файлы (такие как файлы *.exe,* файлы DLL, файлы JAR, сборки и т.п.) или файлы данных, конфигурационные файлы, HTML-документы, файлы с диаграммами и т. д. Чаще всего перечень артефактов внутри узла указывает на то, что на данном узле артефакт разворачивается в запускаемую систему.

Артефакты часто являются реализацией компонентов.

Артефакты можно изображать в виде прямоугольников, так же как и классы, или перечислять их имена внутри узла. Если вы показываете эти элементы в виде прямоугольников, то можете добавить ключевое слово «artifact».

Можно сопровождать узлы и артефакты значениями-метками, чтобы указать различную интересную информацию об узле, например поставщика, операционную систему, местоположение – в общем, все, что важно.

Отношения

Отношение ассоциации между узлами показывает пути коммуникации. Возможно использовать показатели кратности, показывающие сколько узлов принимают участие в отношении.

Отношение зависимости показывает место развёртывания артефактов. Можно просто разместить артефакт внутри узла, либо связать не находящийся внутри узла артефакт с узлом отношением зависимости типа «deploy».

Семинар

1. 20 рабочих станции и сервер БД объединены в локальную сеть. На них установлены ОС Microsoft. Обращение к БД происходит из C++ приложений, развернутых на станциях. На сервере установлена СУБД MS SQL Server.

Нарисовать диаграмму развертывания.

2. В системе имеется следующее оборудование: 10 рабочих станции, взаимодействующих с Интернет через прокси-сервер. Прокси-сервер обеспечивает еще и безопасность. Коммуникация прокси-сервера с внешним миром происходит через провайдера услуг, осуществляющего маршрутизацию к нужному клиентам серверу. В качестве сред выполнения и артефактов использовать следующее ПО: браузер, Web-сервер, ОС Windows и Linux, защитный экран, ОС Unix (у провайдера), файлы данных о клиентах и т.п.

Нарисовать диаграмму развертывания.

Диаграмма развертывания

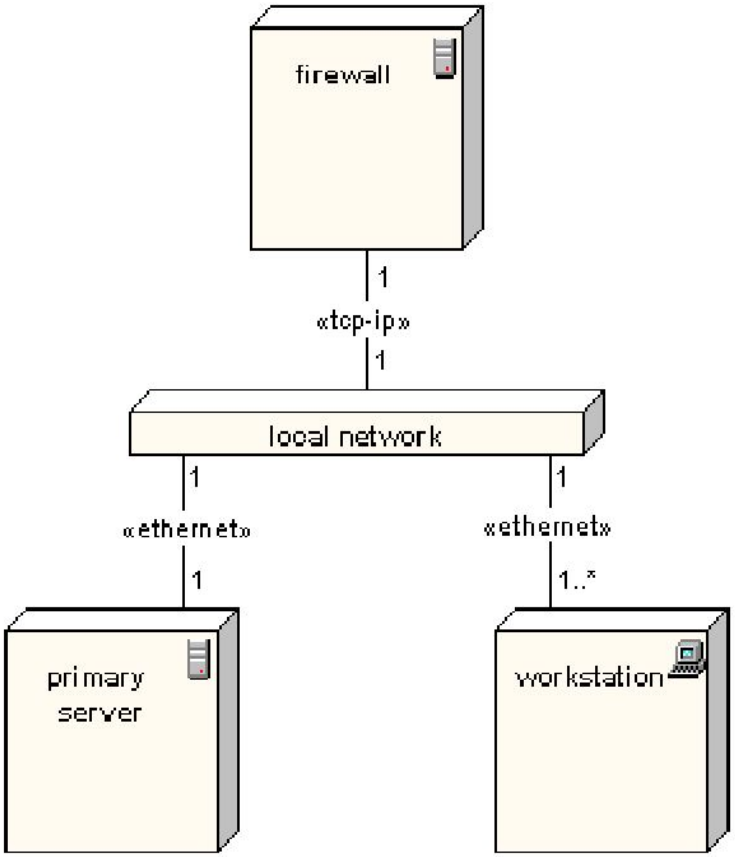
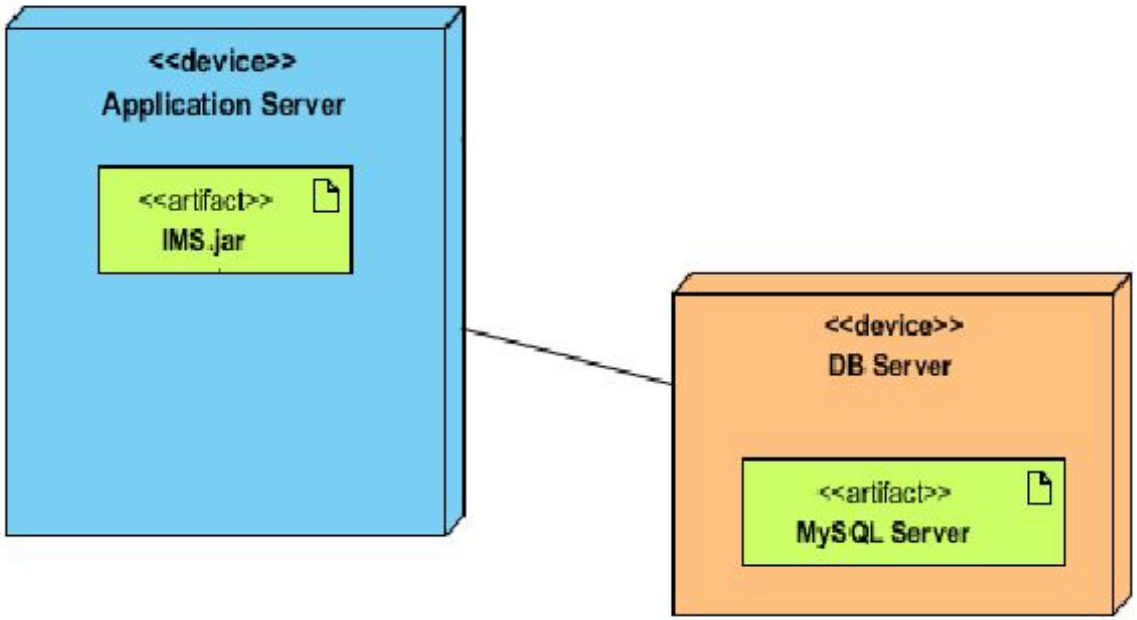
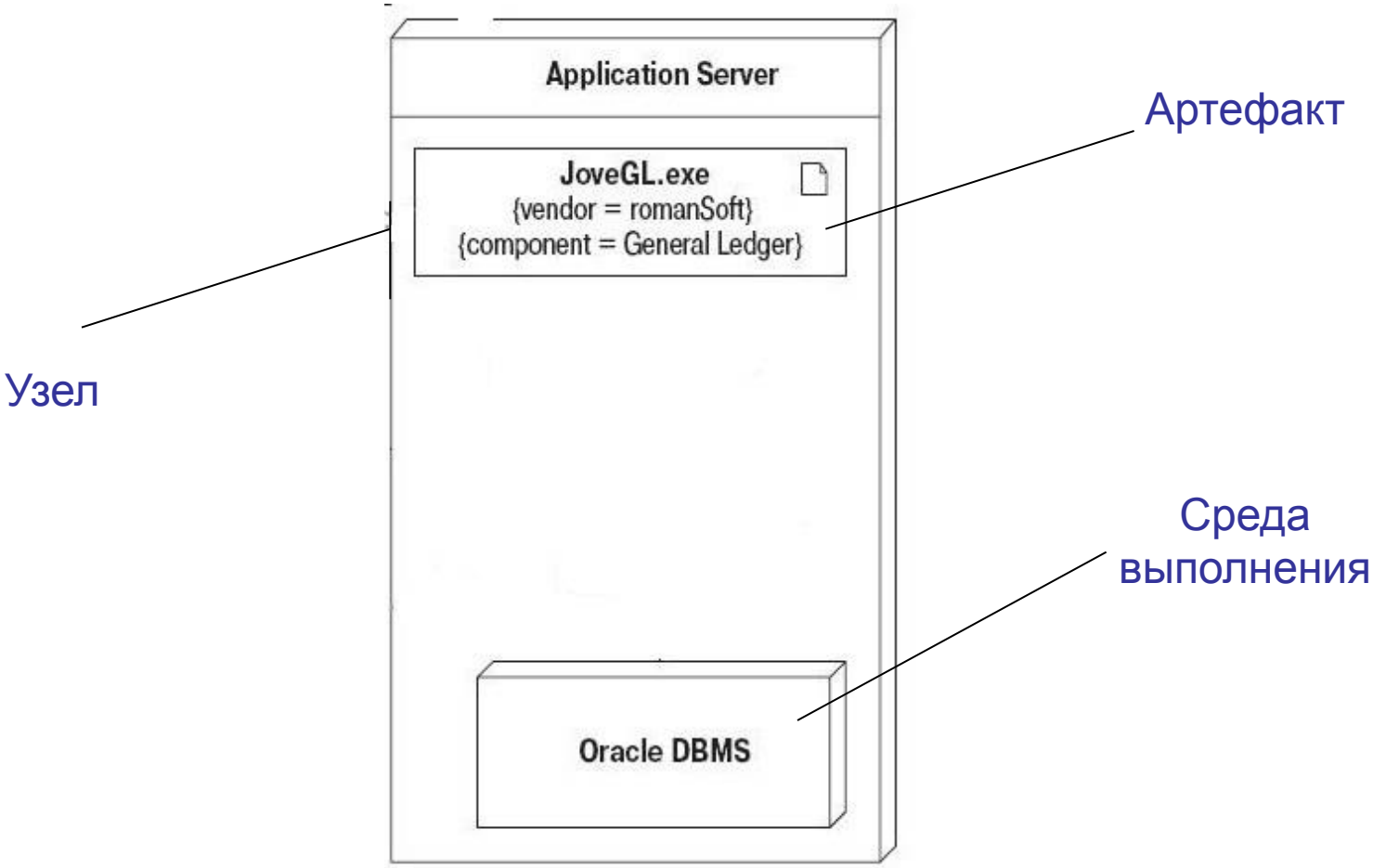


Диаграмма развертывания



Узел


Application Server
JoveGL.exe
{vendor = romanSoft}
{component = General Ledger}
Oracle DBMS
Артефакт
Узел
Среда
выполнения

Тестирование

Тестирование – процесс испытаний или прогонов программной системы с целью обнаружения ошибок.

Задачи:

- Показать, что программная система завершает работу, а не зацикливается
- Сопоставить полученные результаты с ожидаемыми.

Тестовый набор – это входные данные плюс ожидаемые результаты.

Стратегии и методы тестирования

Стратегии

- белого ящика (glass/white box), доступен исходный код:

- выполнение всех операторов;
- покрытие всех линий передачи управления;
- прохождение всех путей от входа к выходу.

- черного ящика (black box), доступна только спецификация:

- метод эквивалентных разбиений;
- метод анализа граничных условий.

Проблема остановки

- В теории вычислимости **проблема остановки**

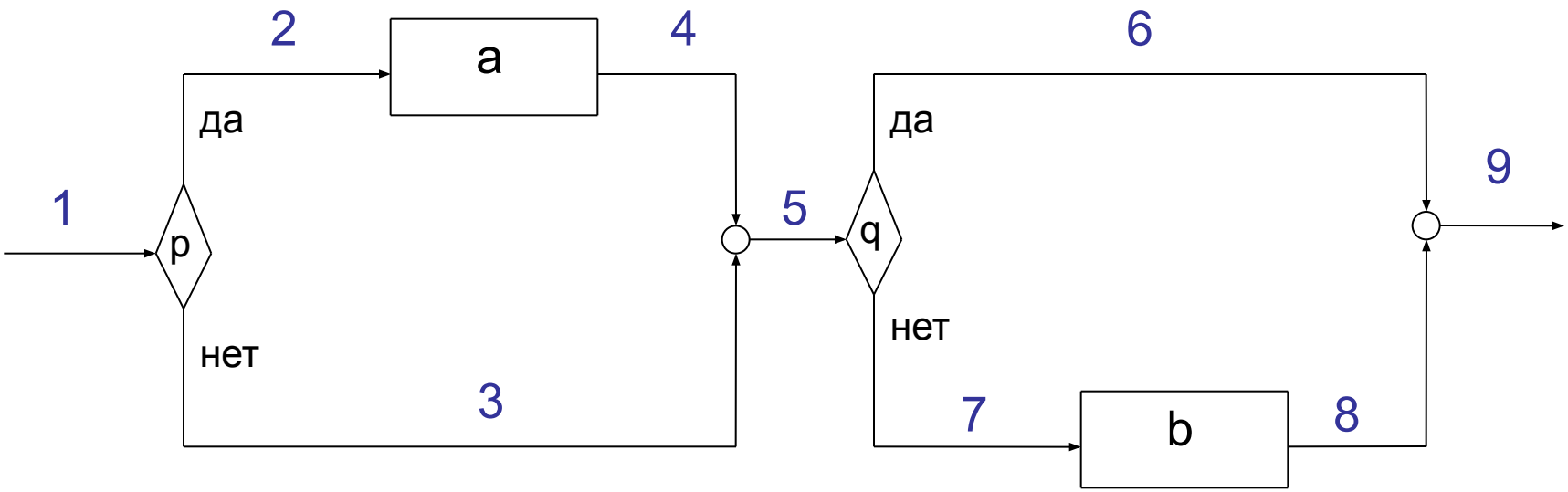
может быть неформально поставлена в виде:

Даны описание алгоритма и его входные данные, требуется определить, сможет ли выполнение алгоритма с этими данными завершиться.

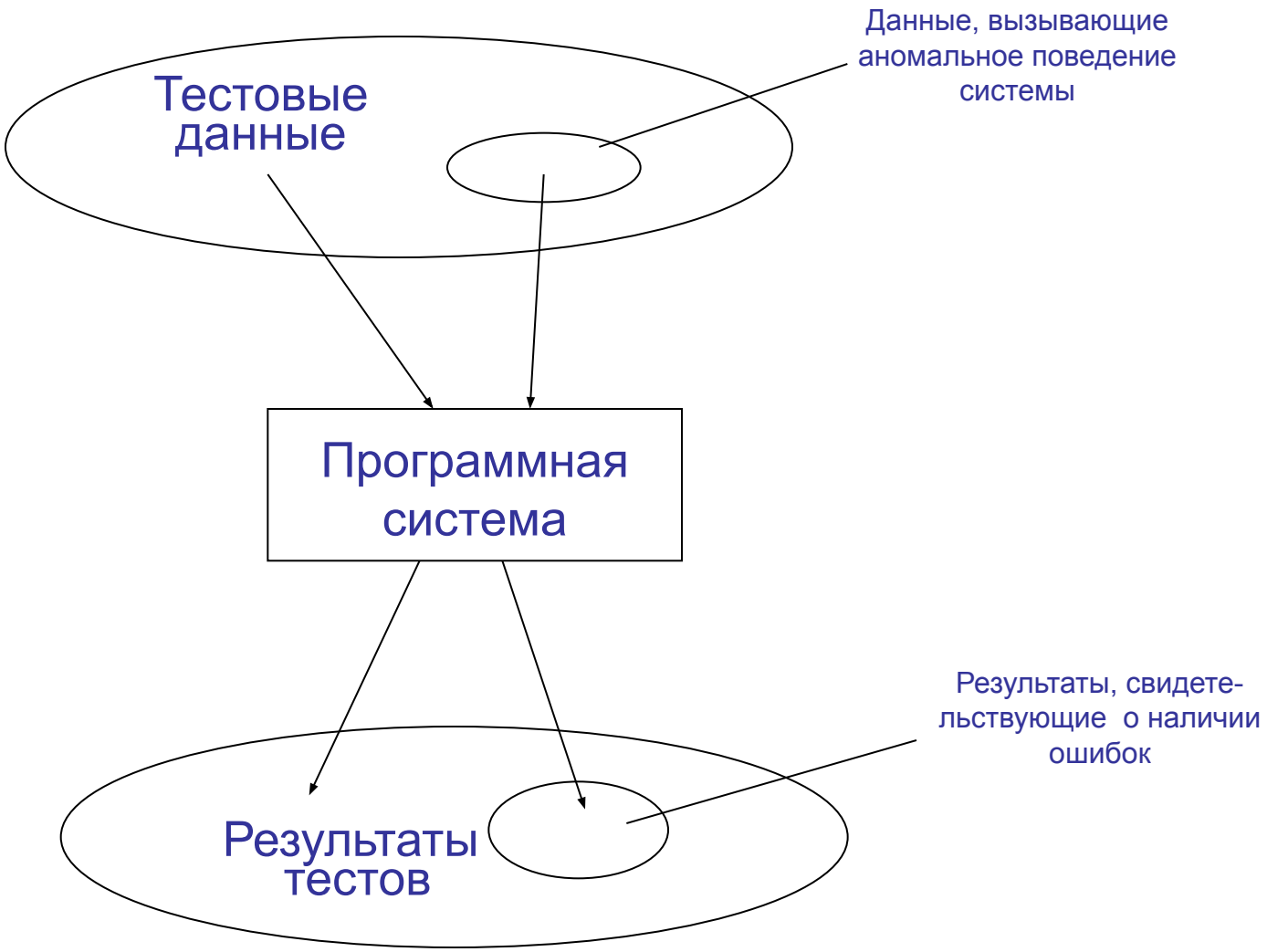
Альтернатива - алгоритм работает всё время без остановки.

- Алан Тьюринг (Великобритания) доказал в 1936 году, что общего алгоритма решения проблемы остановки для любых возможных входных данных не существует. Можно сказать, что проблема остановки неразрешима на машине Тьюринга.
- Петр Новиков (Россия) в 1955 году показал, что проблема остановки эквивалентна так называемой "проблеме тождества слов" в теории групп и компьютерной лингвистике.

Стратегия белого ящика



Стратегия черного ящика

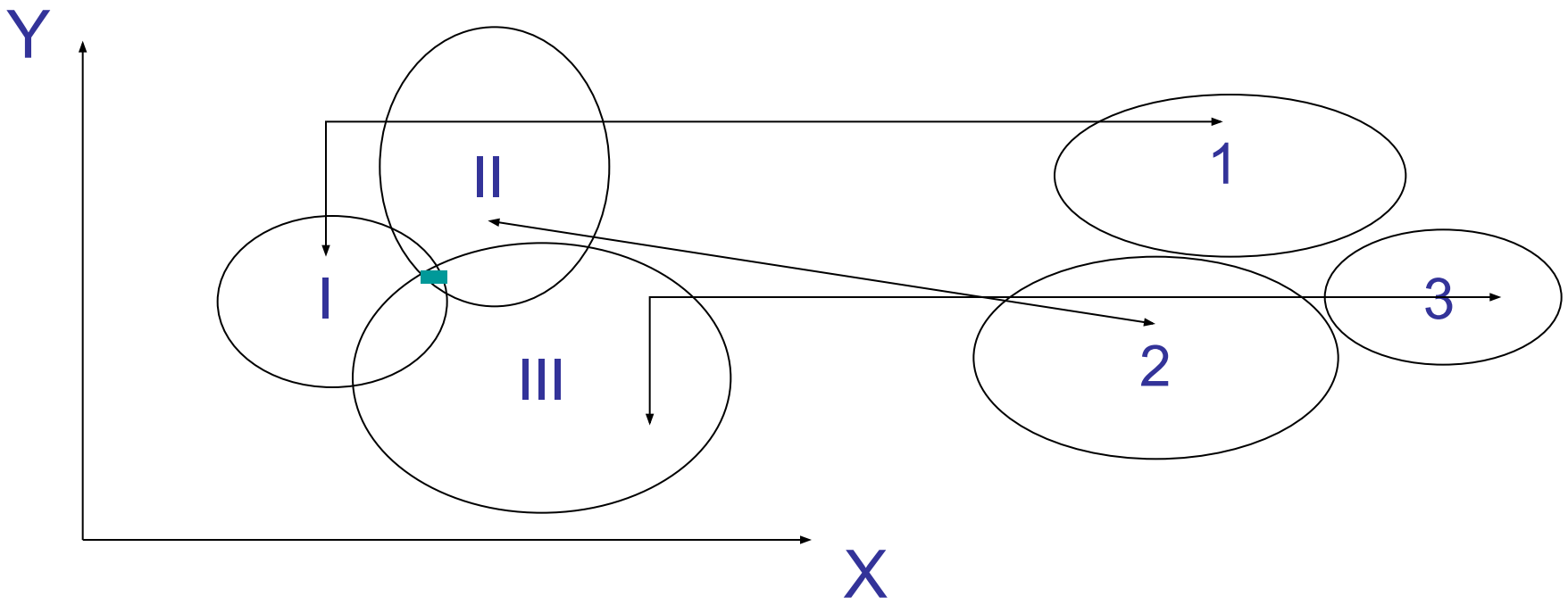


Метод эквивалентных разбиений

Множество потенциально возможных ошибок разбивается на непересекающиеся подмножества.

Область значений входных переменных (тестовых наборов) разбивается на классы эквивалентности.

Каждому классу эквивалентности ставится в соответствие подмножество возможных ошибок.



Классы эквивалентности

Классы эквивалентности выделяются путем выбора каждого входного условия (обычно это предложение или фраза в спецификации) и разбиением его на две или более групп. Предполагается, что для любого значения тестового набора, принадлежащего классу эквивалентности, поведение программы идентично.

Выделяют правильные и неправильные классы эквивалентности. Правильные классы соответствуют входным данным программы, с которыми ПС должна корректно работать; неправильные классы эквивалентности представляют все остальные входные данные.

Метод анализа граничных условий

1. Тестовые наборы принадлежат границам классов эквивалентности или располагаются рядом с границами.

(значения входной переменной X лежит в диапазоне [0,2])

1. Тестовые наборы для максимальных и минимальных значений входных данных.

(число записей во входном файле)

1. Тестовые наборы принадлежат границам области результатов и пограничным областям.

(определение вероятности события)

1. Тестовые наборы для максимальных и минимальных значений выходных данных.

(поисковая система, число релевантных ответов)

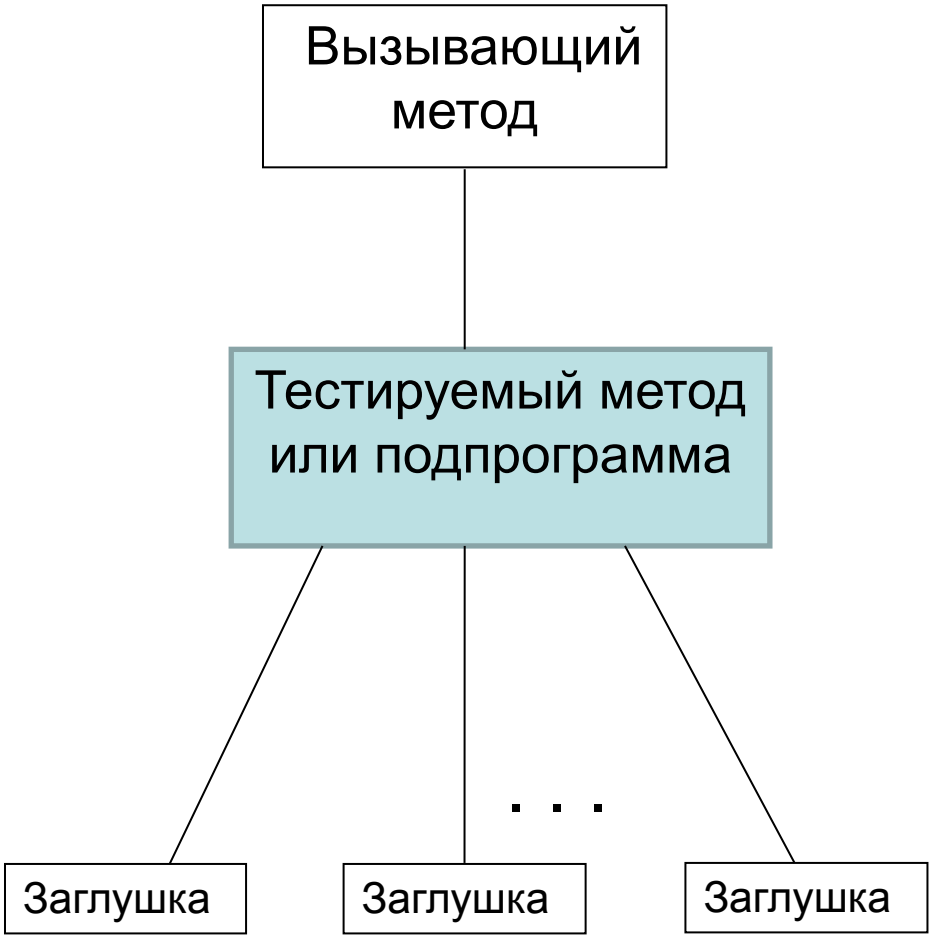
Тестирование подпрограмм и методов

Модульное тестирование предусматривает автономное, отдельное тестирование подпрограммы или метода. Автономное, по крайней мере, по отношению к другим модулям программной системы.

Интеграционное тестирование предусматривает тестирование метода совместно с другими методами этого и/или других модулей.

То есть, для интеграционного тестирования должны быть запрограммированы все модули разрабатываемой программной системы или подсистемы.

Автономное тестирование



Тестирование подпрограмм и методов

Модульное и интеграционное тестирование: сравнение

- *Модульное* тестирование более трудоемко.
- Интеграционное тестирование позволяет охватить больше вариантов, то есть подпрограммы и методы, протестированные ранее, подвергаются дополнительной проверке.
- Модульное тестирование может проводиться одновременно несколькими программистами.
- При интеграционном тестировании раньше обнаруживаются ошибки, возникающие при взаимодействии компонентов программной системы.

Автоматическое тестирование

JUnit – специальная библиотека классов для тестирования программ на Java.

DUnit, NUnit, CPUnit – клоны библиотеки для других языков.

Тестируется работа методов созданного класса. Для этого пишется небольшое приложение, в котором создаются объекты класса, указываются вызываемые методы и формируется условие проверки результатов. Все остальное делается автоматически.

Тестирование программной системы

1. Тестирование интерфейса пользователя.
2. Тестирование на предельных объемах.
3. Тестирование на предельных нагрузках или стрессовое тестирование.
4. Тестирование производительности.
5. Тестирование безопасности.
6. Тестирование требований к памяти.
7. Тестирование совместимости.
8. Тестирование надежности.
9. Тестирование восстановления.
10. Тестирование установки (инсталляции).
11. Конфигурационное тестирование.
12. Регрессионное тестирование.

Тестирование инсталляции

Инсталлятор - это программа, основные функции которой - установка (инсталляция), обновление и удаление (деинсталляция) программного обеспечения. По работе инсталляционного приложения создается первое впечатление о продукте. Именно поэтому тестирование установки - это одна из важнейших задач тестирования.

Являясь обычной программой, инсталлятор обладает рядом особенностей, среди которых стоит отметить следующие:

- Тесное взаимодействие с операционной системой и зависимость от её компонентов - файловой системы, служб и библиотек
- Совместимость компонентов инсталлируемой системы с разными платформами
- Удобство использования: интуитивно понятный интерфейс, навигация, сообщения и подсказки
- Дизайн и стиль инсталляционного приложения

Регрессионное тестирование

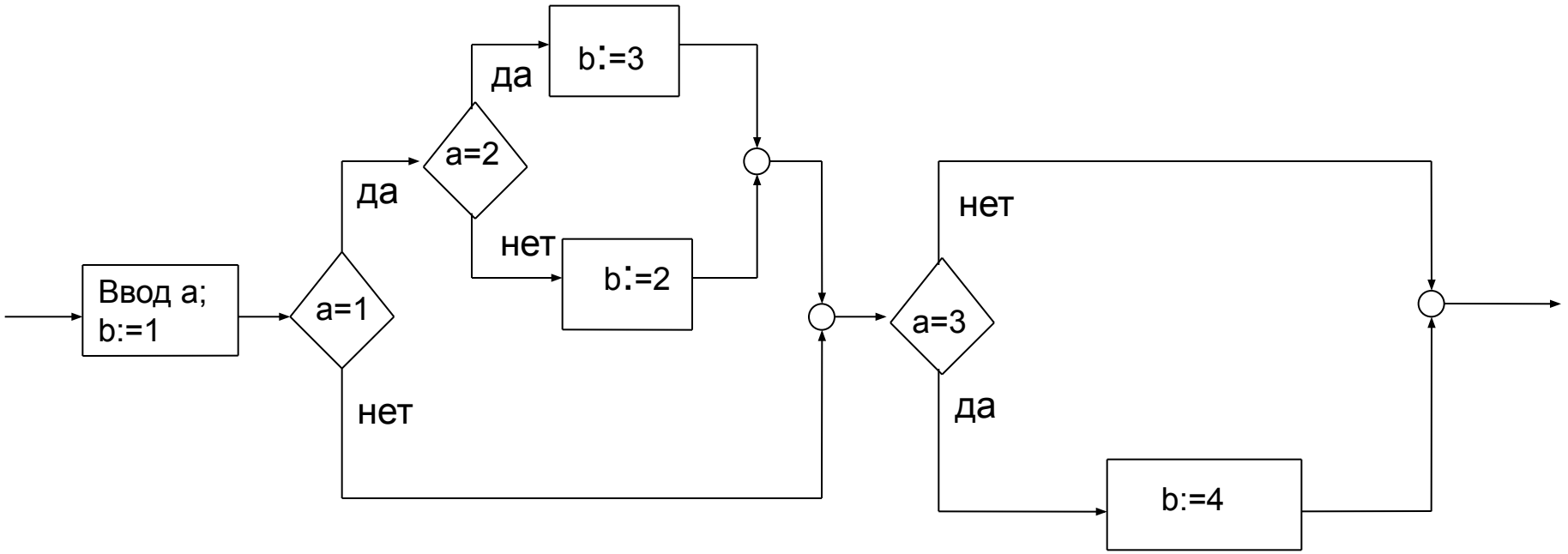
Регрессионное тестирование - это вид тестирования направленный на проверку изменений, сделанных в приложении (исправление ошибки, добавление компонентов, миграция на другую операционную систему или базу данных), для подтверждения того факта, что существующая ранее функциональность работает как и прежде.

Как правило, для регрессионного тестирования используются тесты, написанные на более ранних стадиях разработки и тестирования. Это дает гарантию того, что изменения в новой версии приложения не повредили уже существующую функциональность. Поэтому считается хорошей практикой при исправлении ошибки создать тест именно для неё и регулярно прогонять его при последующих изменениях программы. Хотя регрессионное тестирование может быть выполнено и вручную, но чаще всего это делается с помощью специализированных программ, позволяющих выполнять все регрессионные тесты автоматически. В некоторых проектах даже используются инструменты для автоматического прогона регрессионных тестов через заданный интервал времени.

Семинар №6

1. Используя стратегию белого ящика, подготовить тестовые наборы для программы, структурная схема которой изображена на слайде.
2. Подготовить тестовые данные для программы, определяющей по трем вводимым числам, являются ли они сторонами треугольника, и если да, то к какому типу этот треугольник относится. Использовать методы эквивалентных разбиений и анализа граничных условий.
3. Подготовить тестовые данные для программы, выделяющей цифровые подстроки из произвольной символьной строки. Написать саму программу и осуществить прогон тестов.
4. Подготовить тестовые данные для программы, осуществляющей двоичный поиск в одномерном упорядоченном массиве.

Определение тестовых наборов



Тестовые наборы

(1, 0, 1) (-1, 2, 2) (0.5, 1, -0.9) - не верно

(1, 2, 4) (6, 1.6, 4) (3, 11.4, 8) - не верно

(2, 2, 2) - равносторонний

(2, 3, 3) (4, 3, 4) (9, 9, 10.1) - равнобедренный

(2.5, 3, 4) - разносторонний

(1, 2, 3) - не верно

(2.01, 2, 3) - разносторонний

(1, 0.0001, 1) - равнобедренный

Отладка

Отладка – процесс локализации и исправления ошибок.

В среднем в 1 тысяче строк кода на языке высокого уровня обнаруживают 10-25 ошибок; у Microsoft этот показатель лежит в районе 15. Примерно 80% всех ошибок находится в 20% кода, а 50% ошибок – в 5% кода.

Если в ПО есть ошибки, то причина в 99,9% случаев не в компьютере и компиляторе, а в разработчиках.

Время, затрачиваемое на исправление ошибки:

85% случаев - менее нескольких часов

14% - несколько дней

1% - больше.

Распределение ошибок

Распределение ошибок по этапам разработки:

Анализ требований – 10-25%
Проектирование – 15-30%
Программирование – 45-75%

С ростом объема проекта доля ошибок на этапе программирования снижается.

Возможны ошибке и в самом процессе тестирования. В этом случае реальной ошибки может и не быть.

Типы программных ошибок

- Синтаксические ошибки

Ошибки, связанные с нарушением грамматических правил языка.

Обнаруживаются компилятором.

- Ошибки выполнения

Ошибки, приводящие к ненормальному завершению программы или зацикливанию.

- Логические ошибки

Ошибки, связанные либо с неправильной реализацией алгоритма, либо с неправильно специфицированной функциональностью.

Методы отладки

Методы грубой силы:

- просмотр в узлах
- слежение
- прокрутка

Реализуются с помощью интегрированных в среду разработки программного обеспечения отладчиков

Интеллектуальные методы:

- индукция
- дедукция

Реализуются в процессе тщательного анализа текста программы, часто без использования компьютера

Интеллектуальные методы

Метод индукции:

- определение симптомов ошибки
- определение условий возникновения ошибки
- выдвижение и проверка гипотез о природе ошибки.

Метод дедукции:

- выдвижение гипотез о природе ошибки
- исключение ряда гипотез
- проверка и уточнение выбранных гипотез.

Принципы отладки

1. Избегайте применения только методов грубой силы, чаще используйте интеллектуальные методы отладки.
2. Не зацикливайтесь на поиске одной ошибки (среднее время поиска ошибки в небольших проектах не должно быть больше 1 часа).
3. Не вносите изменений в программу, если ошибка не найдена.
4. Чем больше ошибок обнаружено, тем больше вероятность существования ненайденных ошибок.
5. Ищите ошибку, а не исправляйте ее последствия.
6. Вероятность появления ошибки при внесении изменений выше, чем в старом тексте приложения.
7. Необнаружение ошибок при тестировании с помощью встроенного в среду отладчика не является гарантией их отсутствия.
8. Проводите документирование ошибок.

Исправление последствий ошибок

```
…
Y:=Compute(X);
If X=2 then
   Y:=5.7;
...
```

Инспекции ПО

Инспекции – это процедуры анализа текущих результатов разработки. Могут применяться на разных этапах, чаще всего - на этапах проектирования и тестирования. В качестве текущих результатов рассматриваются подготовленные артефакты: документация в виде текстов и спецификаций, диаграммы, схемы, код. Если инспектируются результаты проектирования, то необходимо привлекать специалистов, хорошо знающих UML. Инспекции, осуществляемые на этапе тестирования, называют сквозными просмотрами программ.

Сквозные просмотры

Состав инспекционной группы на этапе тестирования: разработчик или разработчики, тестировщик, опытный программист, руководитель просмотра (модератор).

Что необходимо иметь для просмотра:

- документация, как минимум, спецификация
- текст программы
- дополнительно список типовых ошибок

Скорость сквозного просмотра ~ 100-150 строк в час.

Прогоны программы как правило не проводятся.

Результаты просмотра:

- список обнаруженных ошибок
- список возможных дефектов.

Сквозные просмотры обычно проводятся более одного раза.

Список вопросов

- Тенденции развития ИТ. Понятие программного обеспечения.
- Рынок ПО в России и в мире. Защита авторских прав разработчиков.
- Обобщенные критерии качества ПО.
- Элементарные критерии качества и метрики ПО.
- Жизненный цикл ПО.
- Технико-экономическое обоснование разработки и техническое задание.
- Функционально-ориентированная стратегия разработки ПО.
- Схема иерархии
- Характеристики модулей
- Объектно-ориентированная стратегия разработки ПО.
- Гибкие технологии разработки ПО
- Риски при разработке ПО.
- Диаграммы прецедентов.

Список вопросов

- Сценарии.
- Этап анализа требований.
- Отношения обобщения и зависимости.
- Отношения между классами: ассоциации.
- Классы-ассоциации.
- Отношение агрегирования.
- Диаграммы объектов.
- Этап объектно-ориентированного проектирования.
- Эволюция в процессе объектно-ориентированной разработки.
- CASE-средства.
- Сопоставление объектно-ориентированной и функционально-ориентированной стратегий разработки ПО.
- Базовые управляющие структуры.
- Декомпозиция структурных схем.
- Диаграммы последовательностей.
- Фреймы в диаграммах последовательностей.

Список вопросов

- Диаграммы деятельности.
- Диаграммы развертывания.
- Рефакторинг.
- Способы организации данных на внешних носителях.
- Организация графического интерфейса.
- Организация интерфейса в Eclipse.
- Тестирование: стратегия белого ящика.
- Тестирование: стратегия черного ящика.
- Тестирование программной системы.
- Регрессионное тестирование
- Модульное и интеграционное тестирование.
- Типы программных ошибок.
- Отладка: методы «грубой силы». Интегрированные отладчики.
- Интеллектуальные методы отладки.
- Принципы отладки.
- Инспекции ПО.