ТЕХНОЛОГИЯ РАЗРАБОТКИ ПРОГРАММ

Начинать важное дело без серьезной подготовки есть первый признак шизофрении.

В.М. Бехтерев

Основы программирования

Содержание

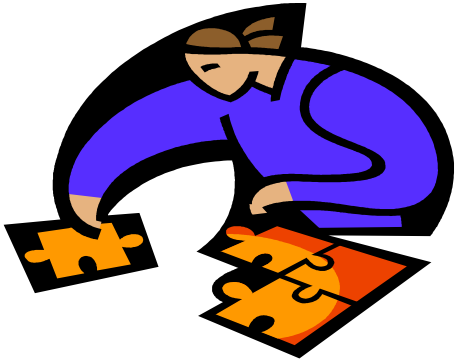
- Технология программирования
- Жизненный цикл ПО
- Пример разработки программы

Технология программирования

- *Технология программирования* – совокупность теории и практической техники, на которые опирается процесс создания программного обеспечения.
- Технология программирования включает в себя
  - *концептуальные средства*, определяющие стиль и методы разработки программ
  - *организационные средства*, определяющие форму труда в команде программистов
  - *программные средства* разработки.

Технология программирования

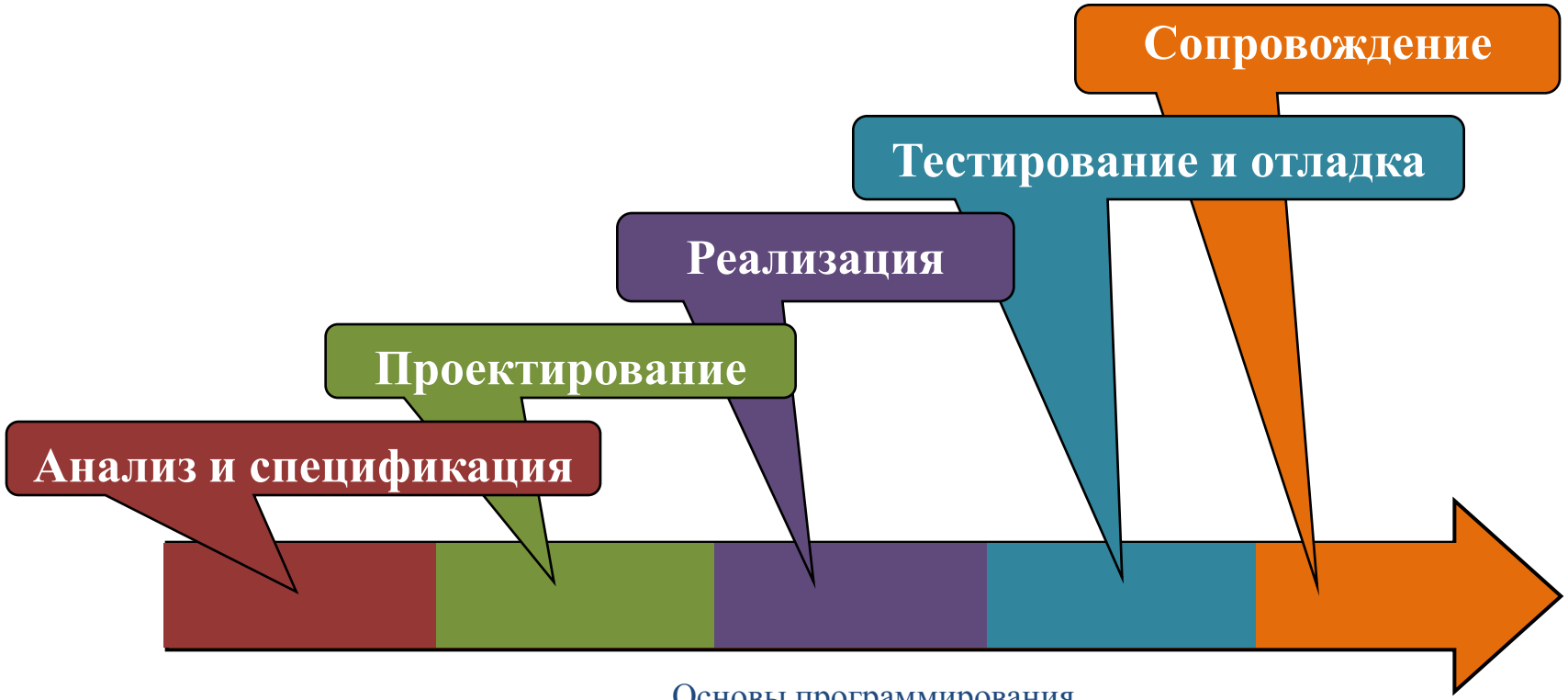
Как решать задачу?



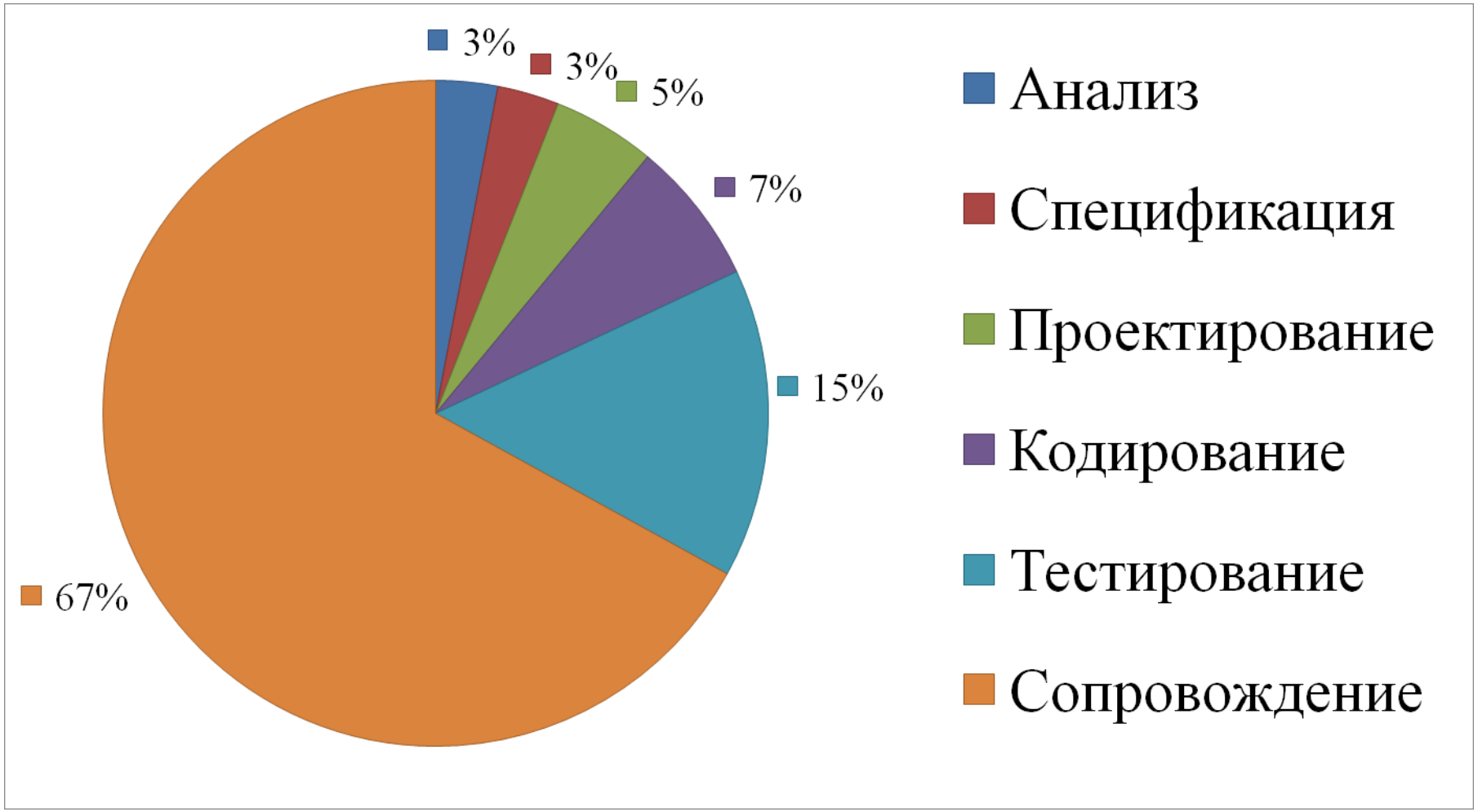
1. Ознакомиться с условием.
2. Составить план решения.
3. Выполнить действия согласно плану.
4. Проверить полученное решение на соответствие заданному условию.
5. Использовать полученный план для решения подобных задач в будущем.

Жизненный цикл ПО

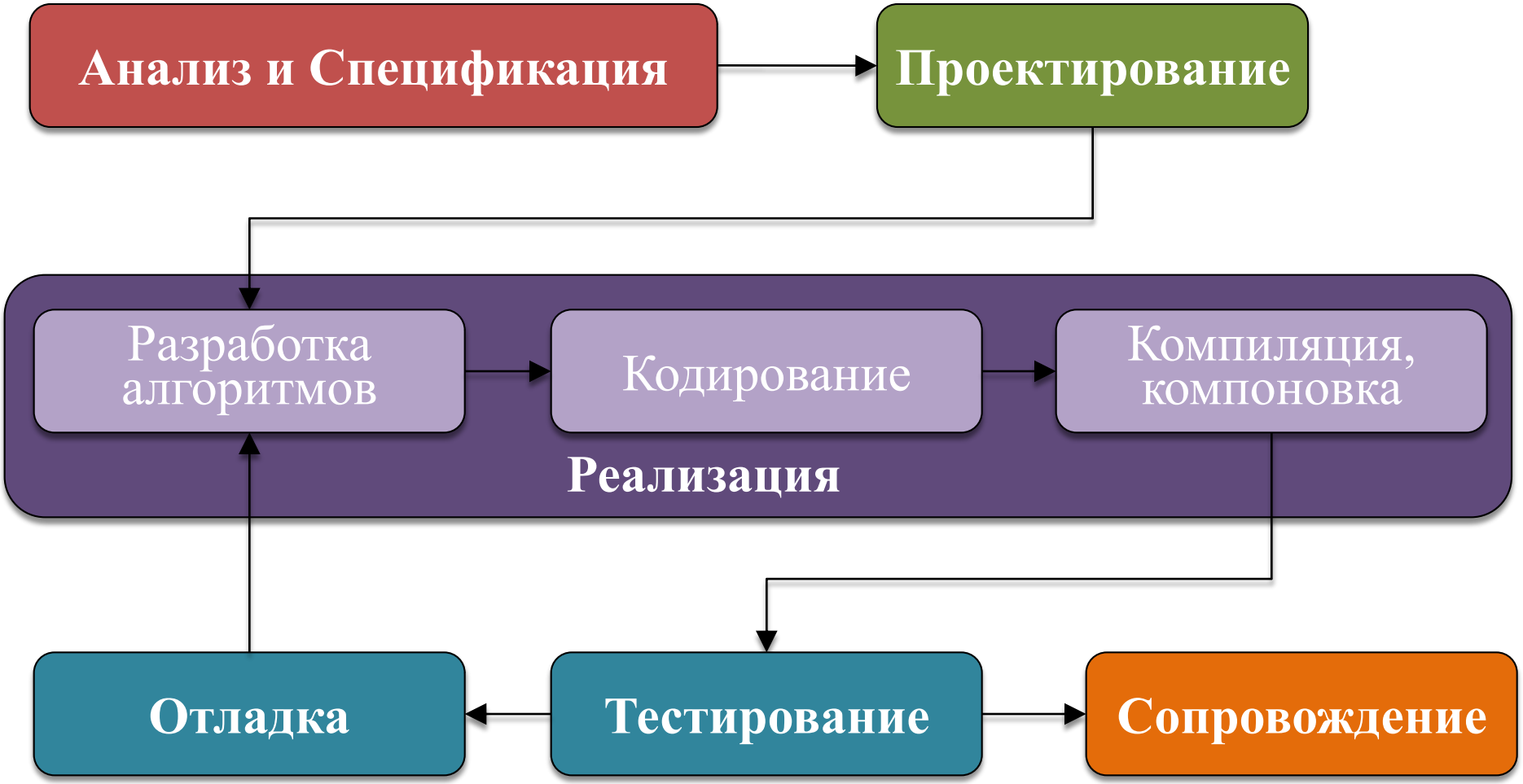
- *Жизненный цикл ПО (software lifecycle)* – период его разработки и эксплуатации, начиная от момента возникновения замысла ПО и заканчивая прекращением всех видов его использования.



Жизненный цикл ПО



Жизненный цикл ПО



Задача

- Если треугольник с заданными сторонами существует, то определить его вид.

Анализ требований

- Программист *совместно* с пользователем устанавливают, *что* должна делать программная система (но *не как* она это должна делать).
  - Трудности перевода: программист и пользователь не являются специалистами в "чужой" области.
  - Пользователь не всегда имеет четкое представление о необходимой ему системе.

Анализ требований: пример

- **Формулировка**
  Вид треугольника – равносторонний, равнобедренный, общий или …?
  Длины сторон – целые или вещественные числа? в каком диапазоне?
- **Входные данные**
  Откуда считываются – с клавиатуры, из (какого?) файла или …?
  Возможен ли ввод некорректных данных (строка символов вместо числа)?
- **Выходные данные**
  Куда записываются – на экран и/или в (какой?) файл или …?
  В каком виде?

Спецификация

- Программист создает *спецификацию* – документ, в котором требования пользователя записываются на достаточно формальном языке.
- Спецификация – единственный документ, по которому определяется правильность результатов разработки.
  - Выбор метода спецификации.
  - Высокая цена возможных ошибок в спецификации.

Спецификация: пример

- **Формулировка**
  Заданы длины сторон треугольника. Если треугольник с такими сторонами существует, определить его вид: равносторонний, равнобедренный, общий. Иначе сообщить, что треугольник не существует.
- **Входные данные**
  Три вещественных числа (не обязательно положительные!). Считываются из текстового файла INPUT.TXT. Всегда корректны.
- **Выходные данные**
  Записываются в текстовый файл OUTPUT.TXT. Одно из сообщений: РАВНОСТОРОННИЙ, РАВНОБЕДРЕННЫЙ, ОБЩЕГО ВИДА, НЕ СУЩЕСТВУЕТ
- **Примеры входных и выходных данных**

INPUT.TXT	OUTPUT.TXT
1 1 1	РАВНОСТОРОННИЙ
1 2 3	НЕ СУЩЕСТВУЕТ
3 4 5	ОБЩЕГО ВИДА

Проектирование

- Разбиение задачи на подзадачи (модули).
  - *Модуль* автономно решает одну подзадачу.
  - Разбиение до достижения определенного уровня простоты.
- Спецификация каждого модуля
  - Назначение.
  - Входные данные.
  - Выходные данные.

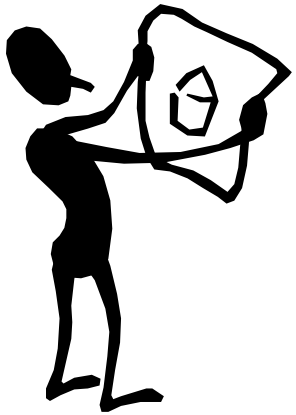
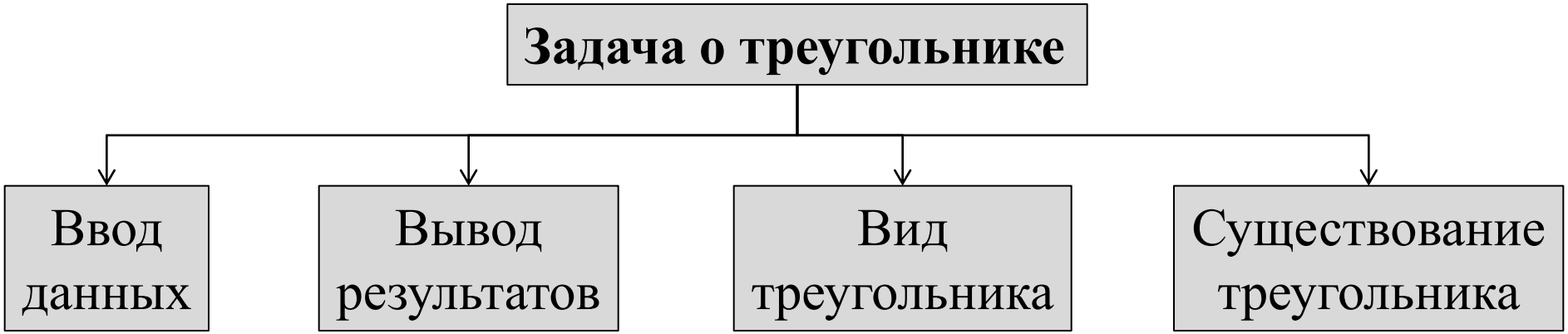


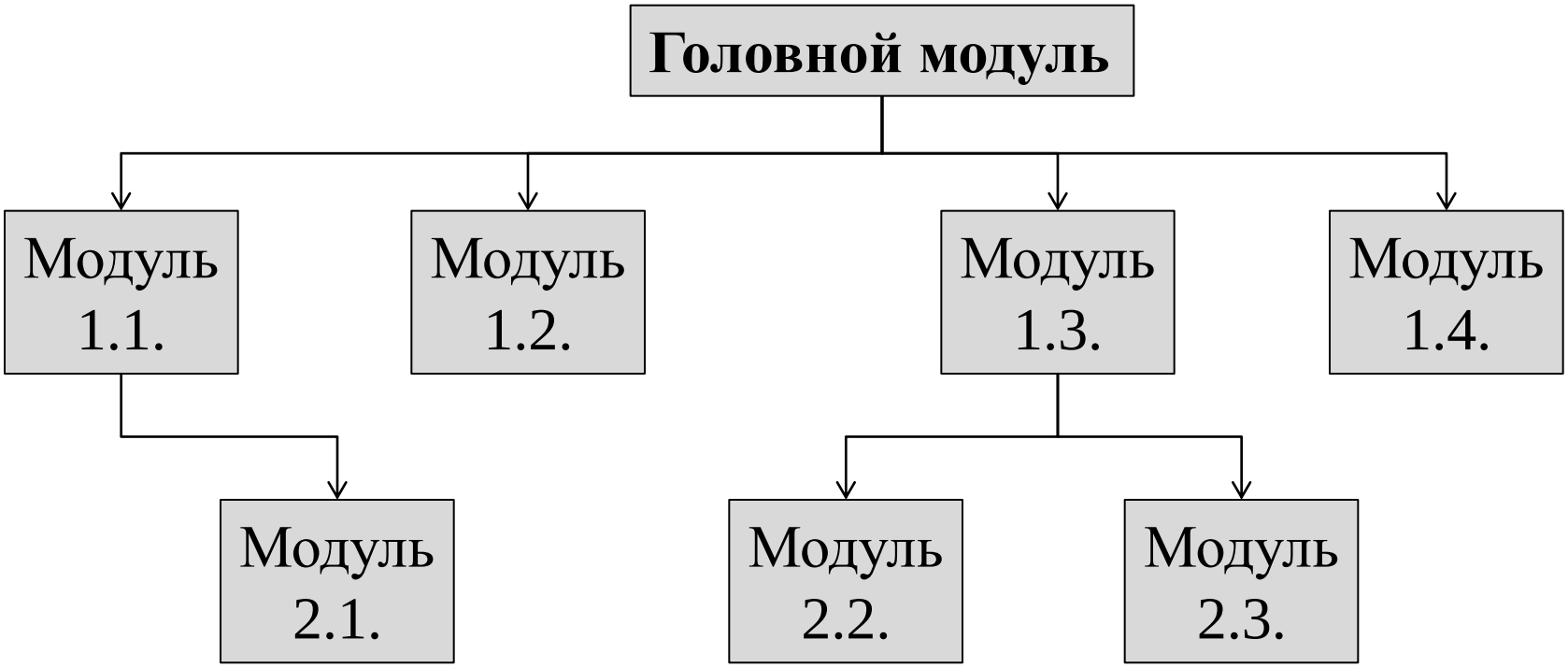
Схема модульной структуры



- В схеме модульной структуры стрелки показывают *связи по управлению* (главный-подчиненный), а не по времени (раньше-позже).
- Изменение реализации подчиненного модуля влечет изменение поведения главного по отношению к нему модуля.

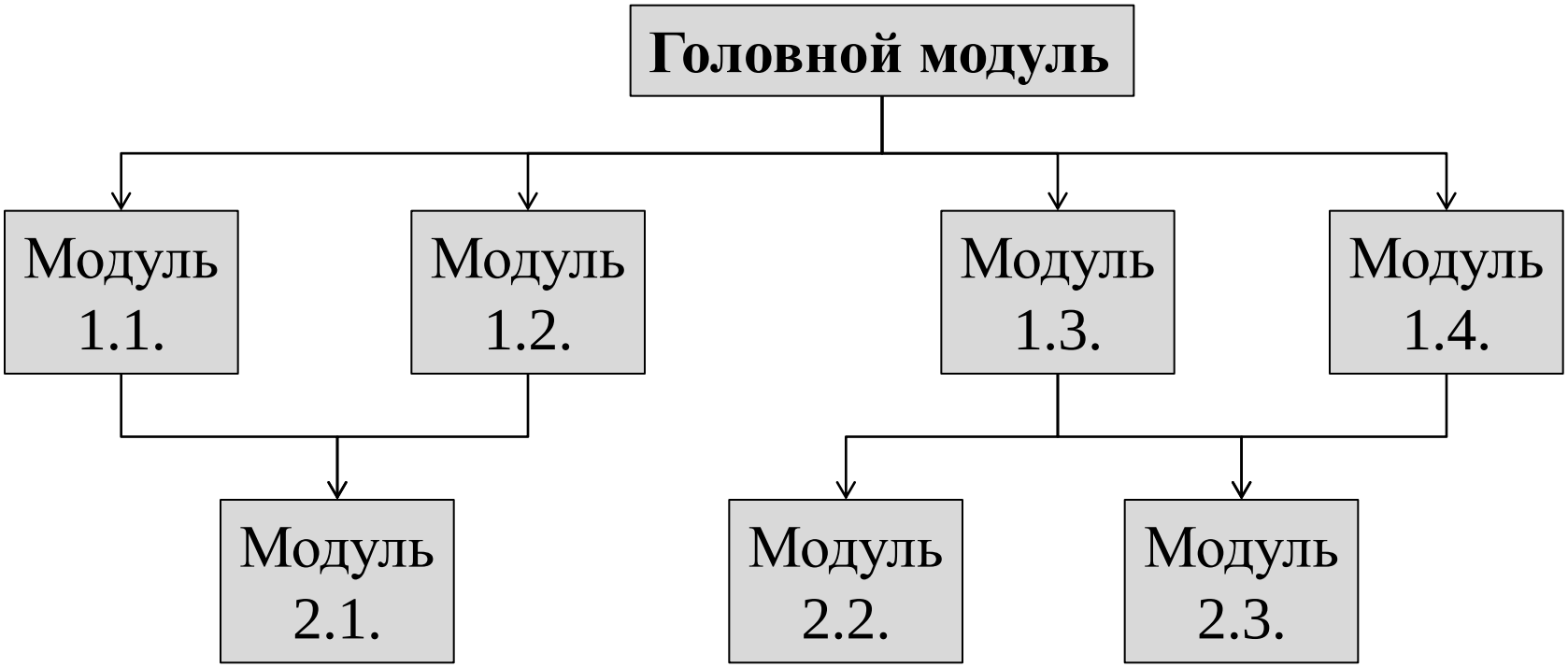
Иерархия модулей

- Сильная иерархия



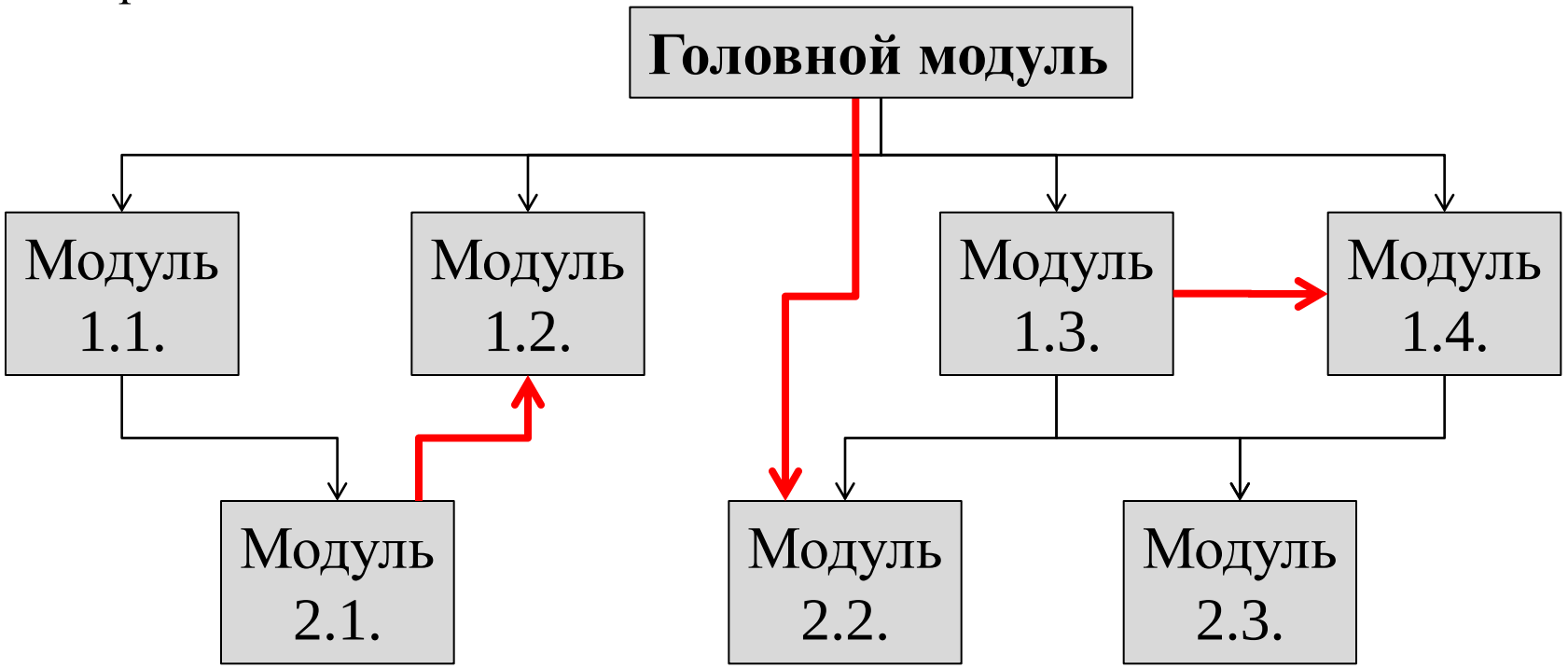
Иерархия модулей

- Ослабленная иерархия



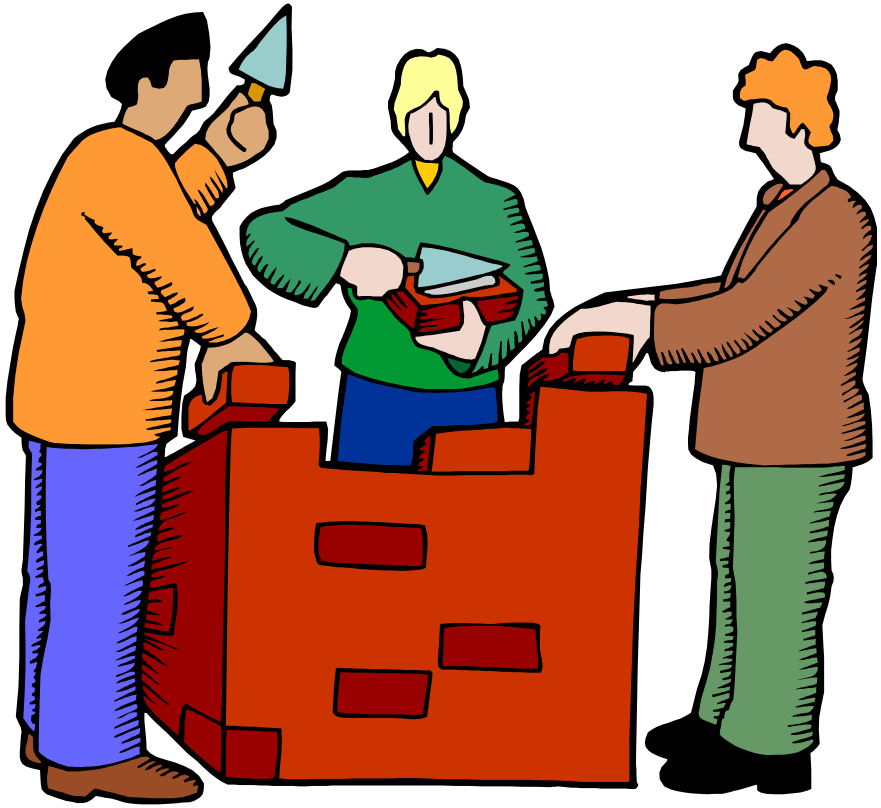
Иерархия модулей

- ***Недопустимые*** межмодульные связи
  - снизу вверх
  - через уровень
  - горизонтальные

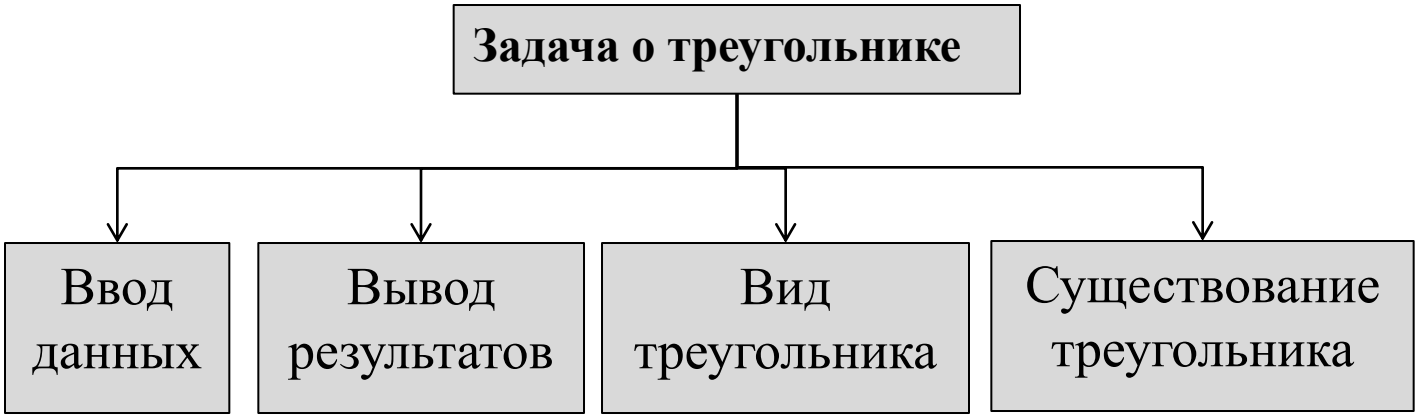


Проектирование: модули

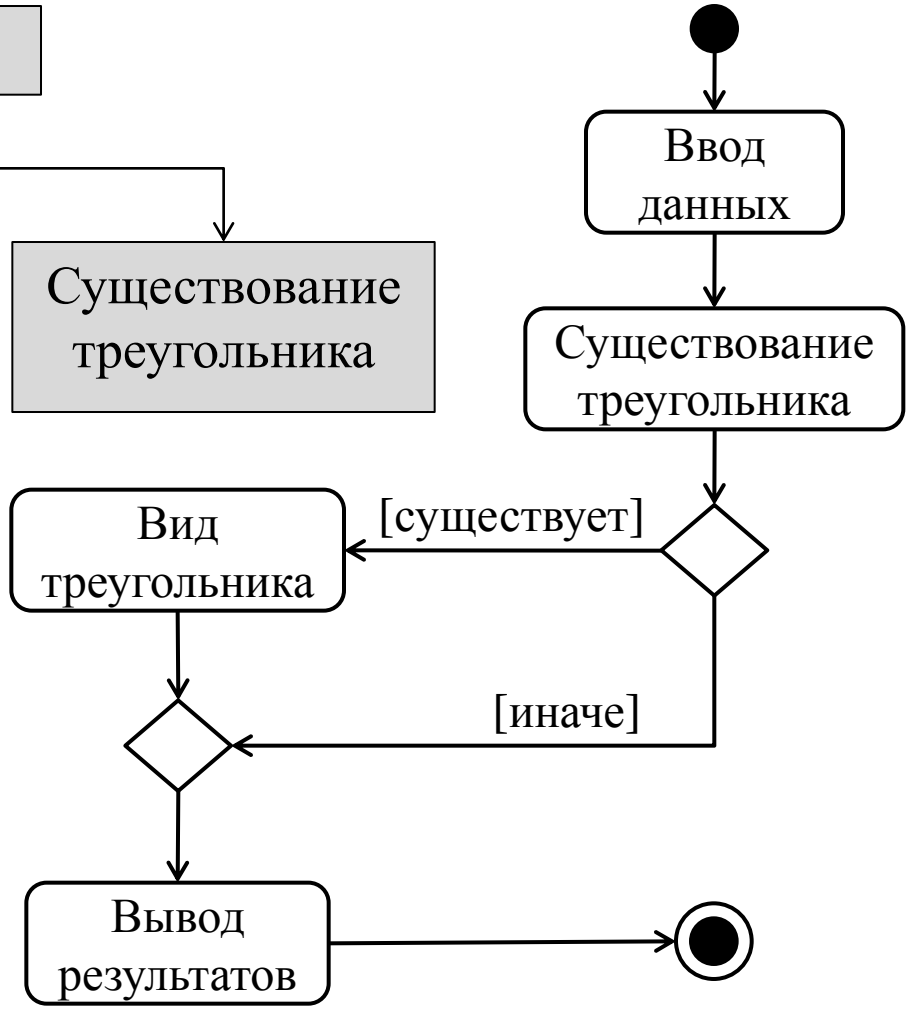
- Модули – "кирпичики" для построения "здания" программы.



Модули

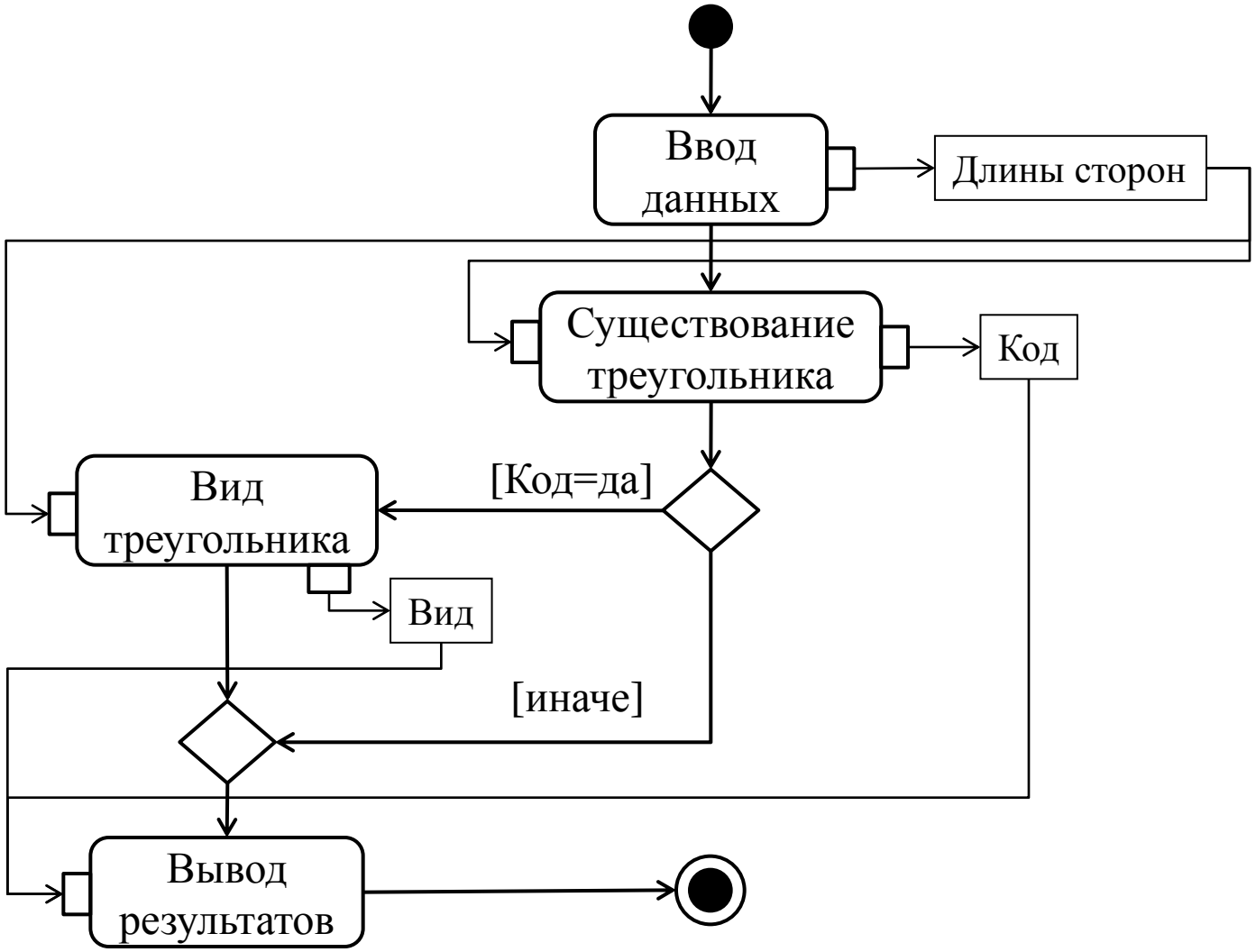


- Композиция спроектированных модулей должна обеспечивать решение задачи.



Взаимодействие модулей

- Передача результатов работы одного модуля в качестве входных данных другого модуля.



Спецификации модулей

- **Модуль**: Ввод данных
  - **Назначение**
    - Запрос у пользователя и ввод длин трех сторон треугольника с клавиатуры.
  - **Входные данные**
    - нет
  - **Результаты**
    - `a вещ`
    - `b вещ`
    - `c вещ`

Спецификации модулей

- **Модуль**: Вывод результатов
  - **Назначение**
    - Вывод на экран сообщения о виде треугольника.
  - **Входные данные**
    - `kind цел`
      - `0` ТРЕУГОЛЬНИК НЕ СУЩЕСТВУЕТ
      - `1` ТРЕУГОЛЬНИК ОБЩЕГО ВИДА
      - `2` ТРЕУГОЛЬНИК РАВНОБЕДРЕННЫЙ
      - `3` ТРЕУГОЛЬНИК РАВНОСТОРОННИЙ
  - **Результаты**
    - нет

Спецификации модулей

- **Модуль**: Существование треугольника
  - **Назначение**
    - Определение существования треугольника по заданным длинам сторон.
  - **Входные данные**
    - `a вещ`
    - `b вещ`
    - `c вещ`
  - **Результаты**
    - `code лог`
      - Если треугольник существует `code=TRUE` иначе `code=FALSE`.

Спецификации модулей

- **Модуль**: Вид треугольника
  - **Назначение**
    - Нахождение вида треугольника по заданным длинам его сторон.
    - Треугольник с заданными сторонами заведомо должен существовать.
  - **Входные данные**
    - `a вещ`
    - `b вещ`
    - `c вещ`
  - **Результаты**
    - `kind цел`
      - `3` треугольник равносторонний
      - `2` треугольник равнобедренный
      - `1` треугольник общего вида

Разработка алгоритмов

- Программы = Алгоритмы + Структуры данных.
- Переход от неформального к формальному существенно неформален.

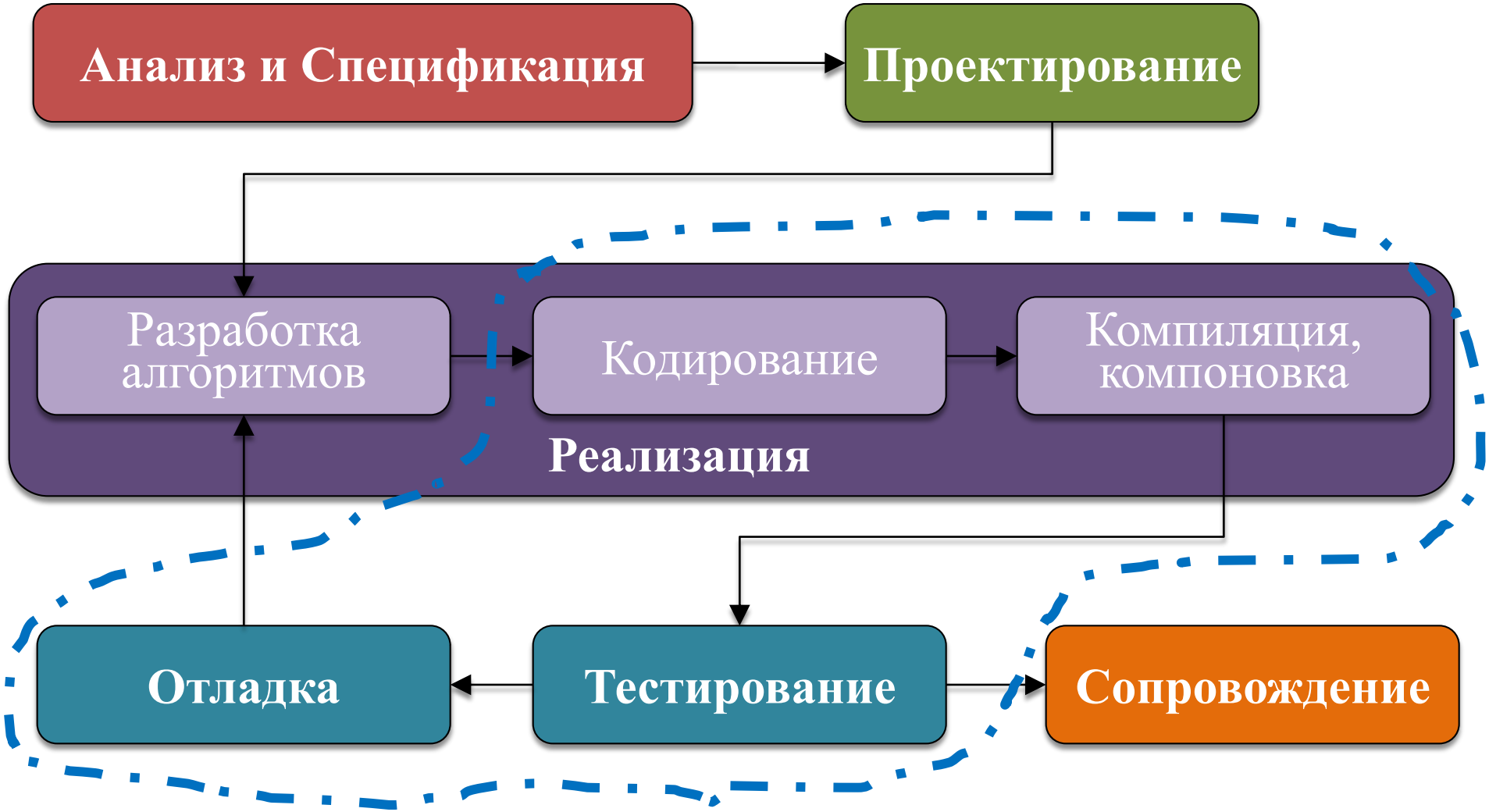
Алгоритмы

- Ввод данных
  - Тривиально.
- Вывод результатов
  - Так же, как Ввод данных.

Алгоритмы

- Вид треугольника
  - Если все стороны равны, то равносторонний. Иначе, если какие-либо две стороны равны, то равнобедренный, иначе общего вида.
  - ```
    если (a=b) и (b=c) то
        РАВНОСТОРОННИЙ
    иначе
        если (a=b) или (a=c) или (b=c) то
            РАВНОБЕДРЕННЫЙ
        иначе
            ОБЩЕГО ВИДА
    ```
    ```

Алгоритмы

- Существование треугольника
  - Если сумма двух любых сторон больше, чем третья сторона, то существует, иначе нет.
  - ```
    если (a+b>c) и (a+c>b) и (b+c>a) то
        TRUE
    иначе
        FALSE
    ```
    ```

Жизненный цикл ПО



Среда программирования

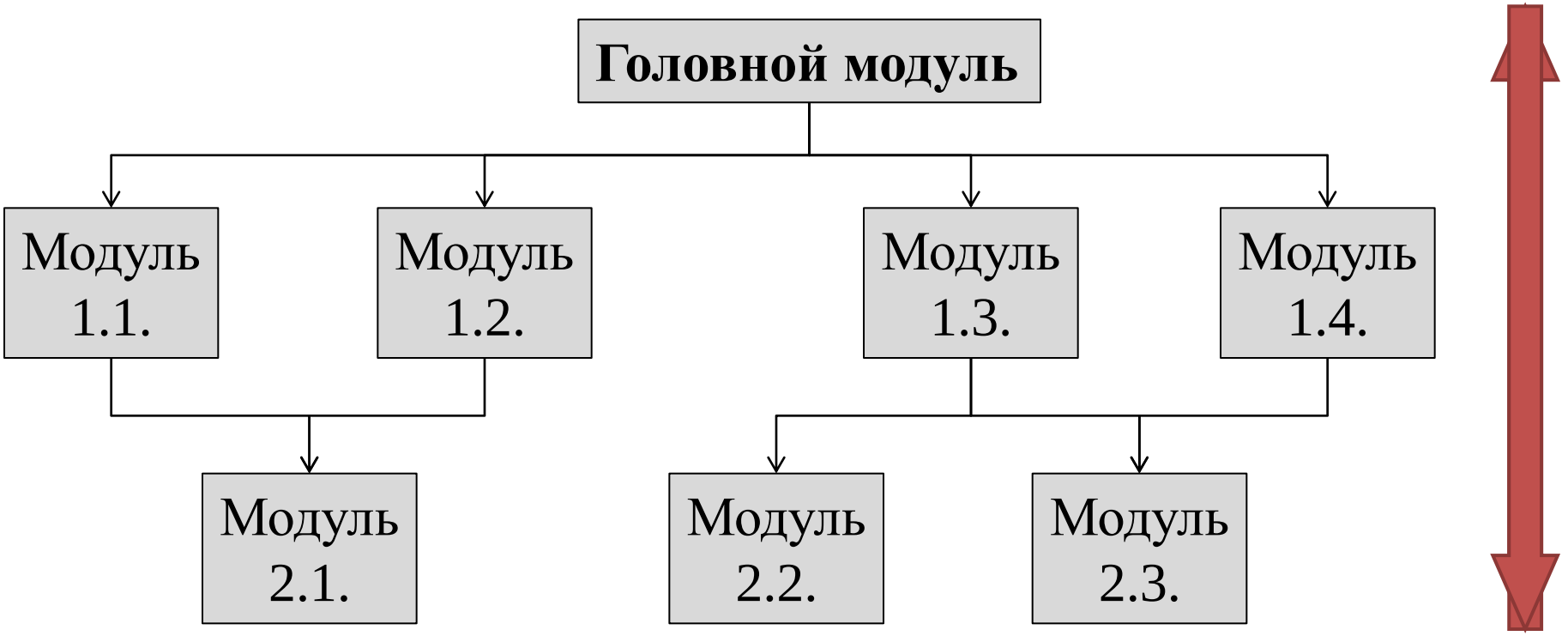
- *Среда (система) программирования* – совокупность инструментов, обеспечивающих преобразование программы на некотором языке программирования в выполнимые вычисления.

Компоненты среды

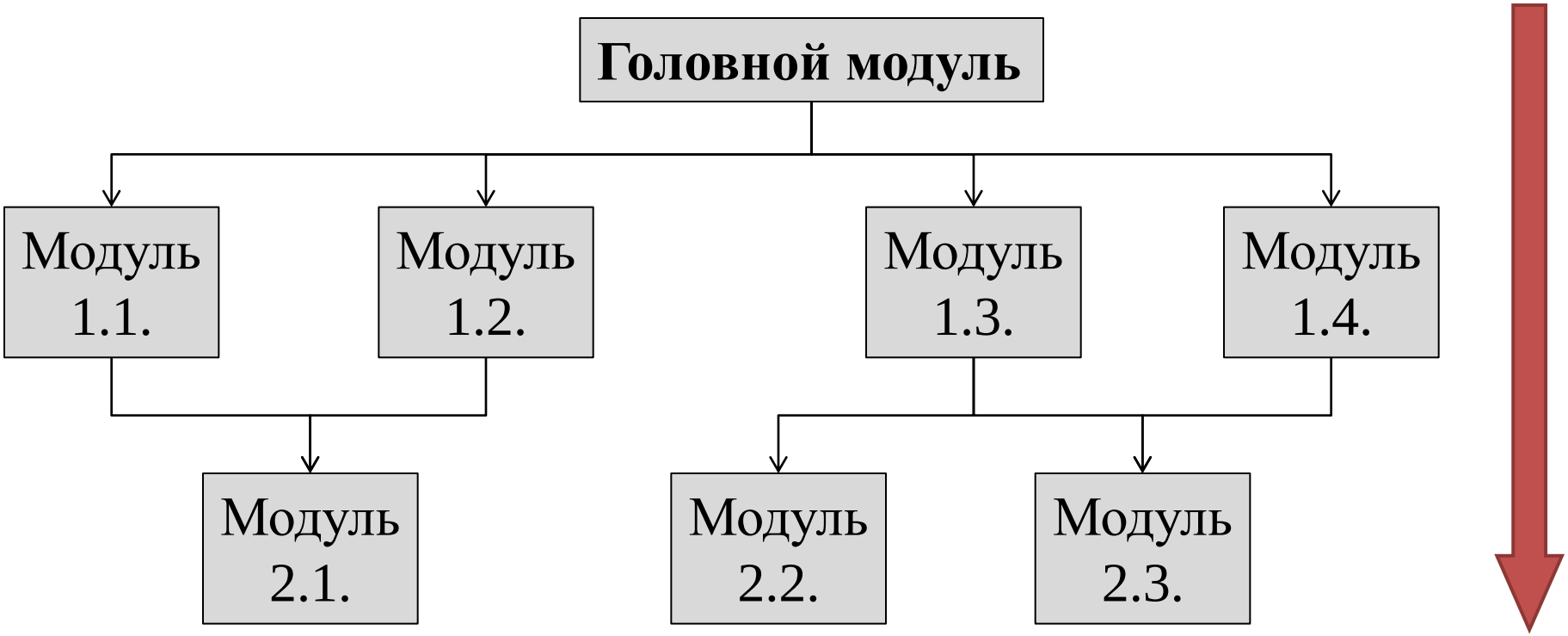
- *Редактор* – средство создания и изменения файлов с исходными с текстами программы.
- *Компилятор* – по файлу с исходным текстом создает *объектный* файл, содержащий команды в машинном коде для конкретного компьютера.
- *Компоновщик* – собирает объектные файлы программы и формирует *исполняемый* файл.
- *Отладчик* – средство управления выполнением исполняемого файла на уровне отдельных операторов программы для диагностики ошибок.

Кодирование

- Создание текстов модулей в виде файлов.
- В каком порядке кодировать модули?

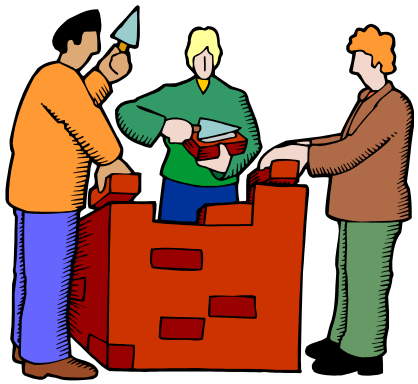


Кодирование – сверху вниз!



Кодирование: подпрограммы

- *Подпрограмма* – это модуль, записанный на определенном языке программирования.
  - *Параметры* подпрограммы соответствуют входным данным и результатам модуля.
  - *Подпрограмма-функция* возвращает одно значение.
  - *Подпрограмма-процедура* не возвращает значений.



Кодирование: головной модуль


```
{ TRIANGLE.PAS
  Нахождение вида треугольника.
  (c) Иванов И.И. (группа ММ-156)
  10-сен-12 }

Program Triangle;

procedure InpData(…

begin
  { Пусто! }
end.
```

Спецификация

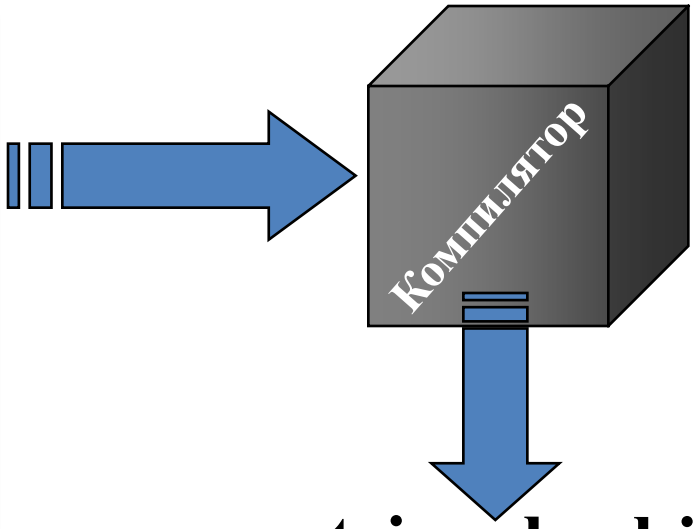
Заголовок

Другие модули

Тело

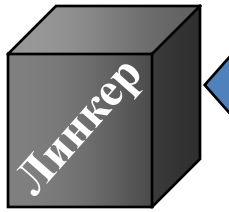
Компиляция и компоновка


```
{ TRIANGLE.PAS
  Нахождение вида треугольника.
  (c) Иванов И.И. (ММ-156)
  10-сен-12 }

Program Triangle;


begin

end.
```



triangle.obj

TRIANGLE_TEXTCODE
DATADATA_BSS
BSSDGROUPSШ
("Ш Н
Ш Н Ъ __UM
_printf qИ y #FИ
p {P _main ZИ
:a§ UЛьh Ъ Г-]-ыЬ
¦V-¶Oa
шTRIANGLE.PAS°
kЄ*¶И ш

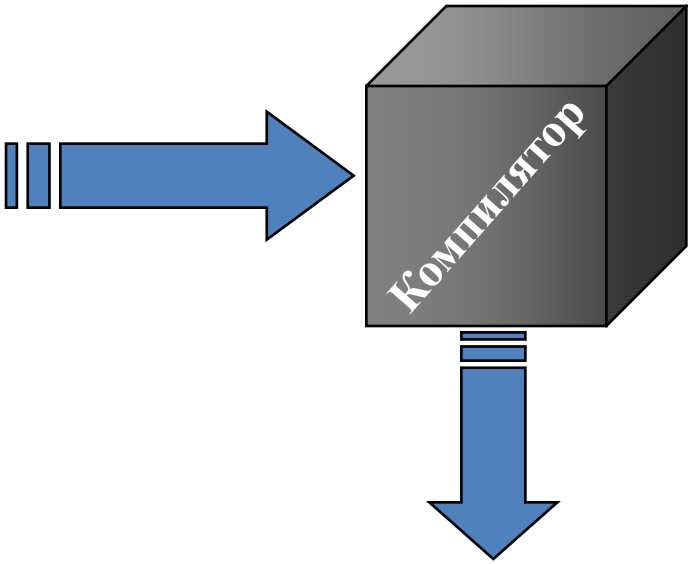


triangle.exe

TRIANGLE_TEXTCODE
+X+ -!sщn¦ -ЙБ ЙГ
+t+@ O++p &¦Зэ.OC+I+rш¦ _
6s _6q _6
ў_Г_0□щ¦d__Й6□+__^]- UЛьVЛv
Vш+_Л¦^]-
Л^Г_$w\A_rWЛFЛN
+}A~ t&¦-Gў+ў+Г+
ыЁ+-ўє6И¶F

Синтаксическая ошибка


```
{ TRIANGLE.PAS
  Нахождение вида треугольника.
  (c) Иванов И.И. (ММ-156)
  10-сен-12 }

Program Triangle;


begin

end
```



Error: unexpected end of file

Здесь нет точки.
Нарушены синтаксические правила языка программирования.

Кодирование модулей

- **Модуль**: Ввод данных
  - **Назначение**
    - Запрос у пользователя и ввод длин трех сторон треугольника с клавиатуры.
  - **Входные данные**
    - нет
  - **Результаты**
    - a вещ
    - b вещ
    - c вещ

```
procedure InpData (
 var a, b, c : Real);

{ Запрос у пользователя и
ввод длин трех сторон
треугольника с клавиатуры.}

begin
   { Пусто! }
end;
```

Заголовок

Спецификация

Тело

Кодирование модулей

- **Модуль**: Вывод результатов
  - **Назначение**
    - Вывод на экран сообщения о виде треугольника.
  - **Входные данные**
    - `kind цел`
      - `0` НЕ СУЩЕСТВУЕТ
      - `1` ОБЩЕГО ВИДА
      - `2` РАВНОБЕДРЕННЫЙ
      - `3` РАВНОСТОРОННИЙ
  - **Результаты**
    - нет

```
procedure OutData (
  K: Integer);

{ Вывод на экран сообщения
о виде треугольника в
зависимости от значения K:
0 ТРЕУГОЛЬНИК НЕ СУЩЕСТВУЕТ
1 ТРЕУГОЛЬНИК ОБЩЕГО ВИДА
2 ТРЕУГОЛЬНИК
РАВНОБЕДРЕННЫЙ
3 ТРЕУГОЛЬНИК
РАВНОСТОРОННИЙ }

begin

end;
```

Кодирование модулей

- **Модуль**: Существование треугольника
  - **Назначение**
    - Определение существования треугольника по заданным длинам сторон.
  - **Входные данные**
    - a вещ
    - b вещ
    - c вещ
  - **Результаты**
    - code лог
      - Если треугольник существует code=TRUE иначе code=FALSE.

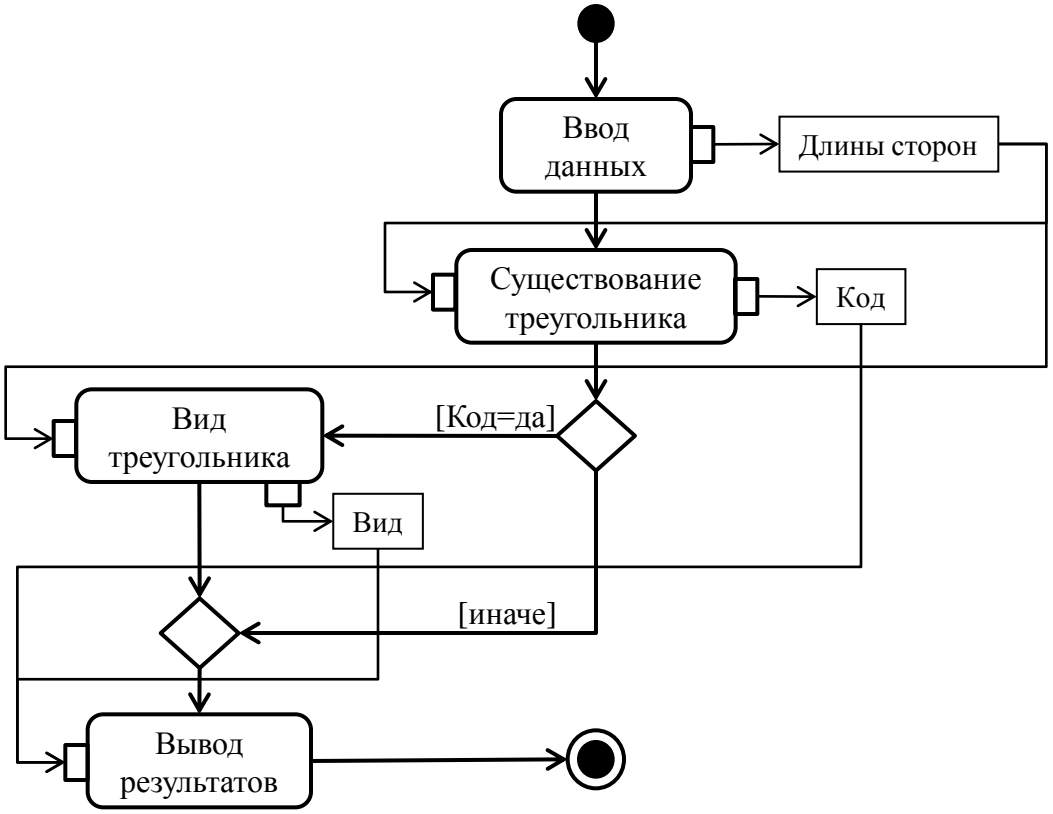
```
function Exist (
 a,b,c: Real):
Boolean;

{ Определение существования
треугольника по заданным
длинам сторон.
Если треугольник
существует, возвращает
TRUE, иначе FALSE.}

begin


end;
```

Кодирование модулей

- **Модуль**: Вид треугольника
  - **Назначение**
    - Нахождение вида треугольника по заданным длинам его сторон.
    - Треугольник с заданными сторонами заведомо должен существовать.
  - **Входные данные**
    - `a вещ`
    - `b вещ`
    - `c вещ`
  - **Результаты**
    - `kind цел`
      - `3` треугольник равносторонний
      - `2` треугольник равнобедренный
      - `1` треугольник общего вида

```
function Kind(
 a,b,c: Real): Integer;

{ Определение вида
треугольника по заданным
длинам сторон.
Возвращает:
3 треугольник равносторонний
2 треугольник равнобедренный
1 треугольник общего вида
Треугольник с заданными
сторонами заведомо должен
существовать. }

begin

end;
```

Кодирование: переменные




```
{ TRIANGLE.PAS
  Нахождение вида треугольника.
  (c) Иванов И.И.
  10-сен-12 }
Program Triangle;

procedure InpData(…

var
  a,b,c: Real;
  Code: Boolean;
  Kind: Integer;

begin

end.
```

Переменные

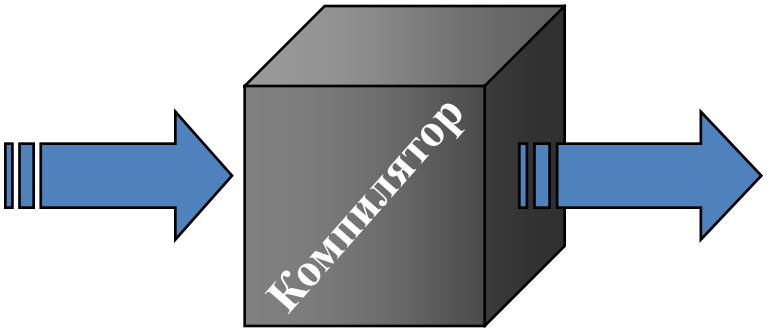
Кодирование: головной модуль


```
{ TRIANGLE.PAS
  Нахождение вида треугольника.
  (c) Иванов И.И.
  10-сен-12 }
Program Triangle;

procedure InpData(…

var
 a,b,c: Real;
 Code: Boolean;
 K: Integer;
```

```
begin
  InpData(a,b,c);
  Code:=Exist(a,b,c);
  if Code=TRUE then
    K:=Kind(a,b,c)
  else
    K:=0;
  OutData(K);
end.
```

Промежуточная компиляция и запуск


```
{…}
Program Triangle;
procedure InpData(…); {…}
begin {Пусто!} end;
procedure OutData(…);{…}
begin   {Пусто!} end;
function Kind(…): …; {…}
begin   {Пусто!} end;
function Exist(…): …; {…}
begin   {Пусто!} end;
var …;
begin
   InpData(a,b,c);
   Code:=Exist(a,b,c);
   if Code=TRUE then
      K:=Kind(a,b,c)
   else
      K:=0;
   OutData(K);
end.
```



- Необходимо убедиться, что головной модуль работоспособен, а затем кодировать подчиненные модули (сверху вниз).

Кодирование модулей


```
procedure InpData (var a, b, c : Real);
{ Запрос у пользователя и ввод длин трех сторон
треугольника с клавиатуры.}
begin
  Write('Введите длины сторон треугольника: ');
  ReadLn(a,b,c);
end;
```

Кодирование модулей


```
procedure OutData(K: Integer);
{ Вывод на экран сообщения о виде треугольника в
зависимости от значения K:
0 ТРЕУГОЛЬНИК НЕ СУЩЕСТВУЕТ
1 ТРЕУГОЛЬНИК ОБЩЕГО ВИДА
2 ТРЕУГОЛЬНИК РАВНОБЕДРЕННЫЙ
3 ТРЕУГОЛЬНИК РАВНОСТОРОННИЙ }
begin
   case K of
   0: WriteLn('ТРЕУГОЛЬНИК НЕ СУЩЕСТВУЕТ');
   1: WriteLn('ТРЕУГОЛЬНИК ОБЩЕГО ВИДА');
   2: WriteLn('ТРЕУГОЛЬНИК РАВНОБЕДРЕННЫЙ');
   3: WriteLn('ТРЕУГОЛЬНИК РАВНОСТОРОННИЙ');
   end;
end;
```

Кодирование модулей


```
function Exist(a,b,c: Real): Boolean;
{ Определение существования треугольника по заданным
длинам сторон. Если треугольник существует,
возвращает TRUE, иначе FALSE.}
begin
  if (a+b>c) and (a+c>b) and (b+c>a) then
    Result:=TRUE
  else
    Result:=FALSE;
end;
```

```
если (a+b>c) и (a+c>b) и (b+c>a) то
   TRUE
иначе
   FALSE
```

Кодирование модулей


```
function Kind(a,b,c: Real): Integer;
{ Определение вида треугольника по заданным длинам сторон.
Возвращает:
3 равносторонний
2 равнобедренный
1 общего вида
Треугольник с заданными
сторонами заведомо
должен существовать. }

begin
   if (a=b) and (b=c) then
      Result:=3
   else
      if (a=b) or (a=c) or (b=c) then
         Result:=2
      else
         Result:=1;
end;
```

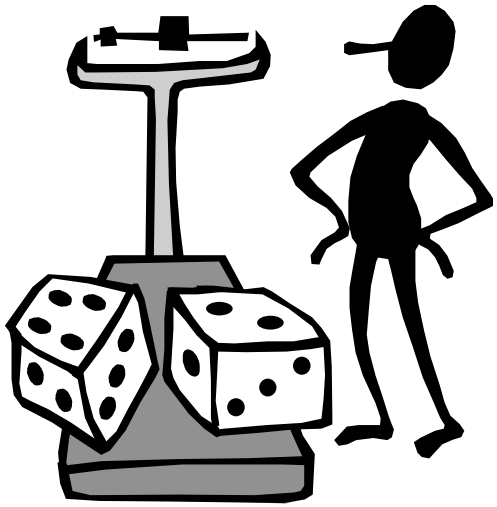
```
если (a=b) и (b=c) то
    РАВНОСТОРОННИЙ
иначе
    если (a=b) или (a=c) или (b=c) то
        РАВНОБЕДРЕННЫЙ
    иначе
        ОБЩЕГО ВИДА
```

Кодирование: "лесенка"

- *"Лесенка"* – отступы от левого края текста, которые показывают структурную вложенность конструкций программы.
  - "Лесенка" не нужна компилятору:
    - можно компилировать программу, текст которой записан в 1 строку.
  - "Лесенка" нужна программисту:
    - ясный и читаемый текст программы.

Тестирование

- Разработчик выполняет программу на *заранее подготовленном им самим* наборе данных (*тесте*), для которого результат работы программы известен .
- *Цель тестирования* – не доказать правильность системы (это невозможно!), а продемонстрировать факт наличия в ней ошибки.



Тестирование

№	Входные данные			Ожидаемый результат	Действительный результат	Тест пройден?
	a	b	c			
1	1	1	1	РАВНОСТОРОННИЙ	РАВНОСТОРОННИЙ	✓
2	1	2	3	НЕ СУЩЕСТВУЕТ	ОБЩЕГО ВИДА	✗
3	3	4	5	ОБЩЕГО ВИДА	ОБЩЕГО ВИДА	✓
4	0	0	0	НЕ СУЩЕСТВУЕТ	РАВНОСТОРОННИЙ	✗
5	4	4	5	РАВНОБЕДРЕННЫЙ	РАВНОБЕДРЕННЫЙ	✓
6	2	1	1	НЕ СУЩЕСТВУЕТ	РАВНОБЕДРЕННЫЙ	✗
7	-1	-1	-1	НЕ СУЩЕСТВУЕТ	Runtime error 106 at 0BEB:0026.	✗

Ошибка времени выполнения

- *Ошибка времени выполнения (run-time error)* происходит при выполнении синтаксически верной программы, когда она производит какое-либо недопустимое действие (деление на ноль и др.).

calc.pas

```
{ Вычисления.
 (c) Иванов И. }
Program Calculation;
var A, B, C: Real;
begin
   Write('Введите A и B: ');
   ReadLn(A, B);

   C:=(A*A+B*B)/(A-B);

   WriteLn('C=', C:5:5);
end.
```

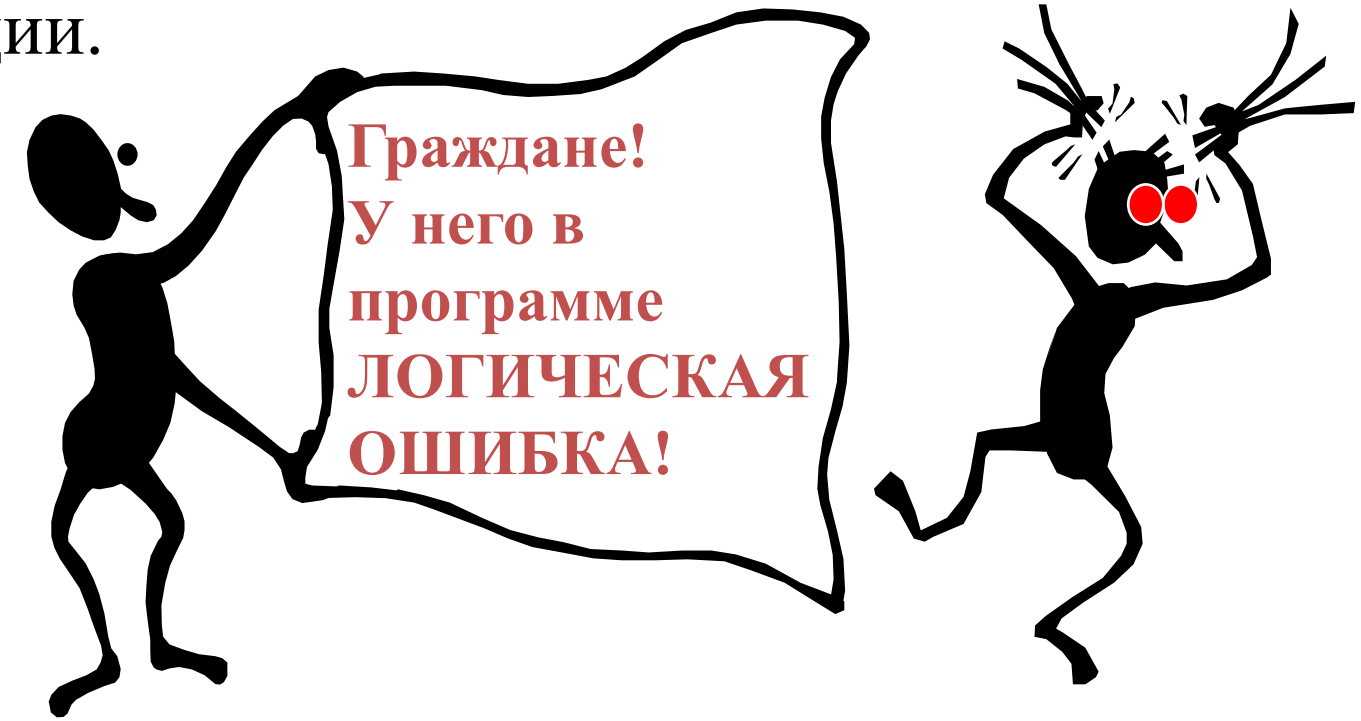
Здесь будет деление на 0 при A=B!

calc.exe

Введите A и B: 1 1
Run-time error 200
at адрес памяти**.**

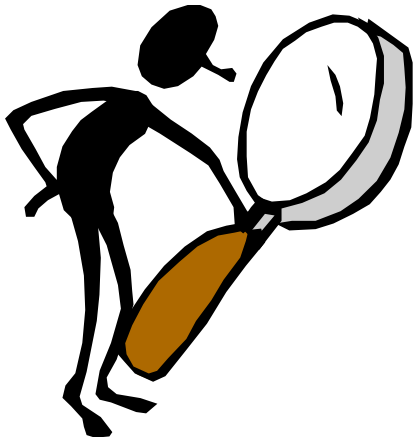
Логическая ошибка

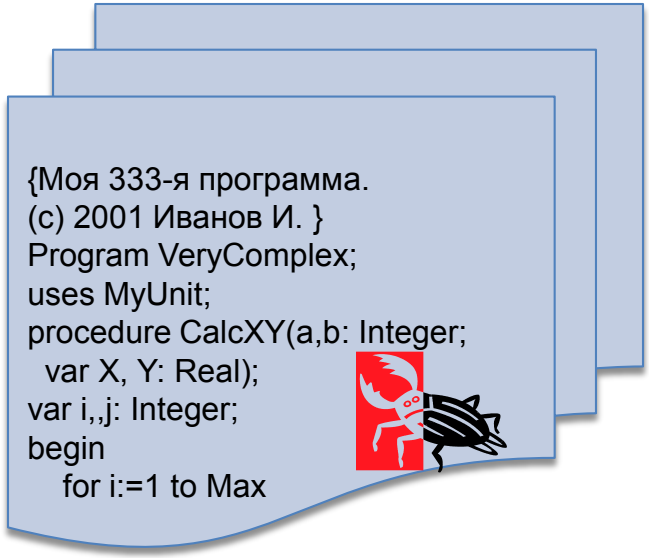
- *Логическая ошибка* – ошибка разработки алгоритма, которая заключается в том, что при выполнении программа (модуль) делает не то, что указано в спецификации.



Отладка

- *Отладка (Debugging)* – процесс поиска и исправления ошибок, выявленных во время тестирования программы.
- *«Сухая» отладка* – по листингу (тексту) программы, без использования компьютера.



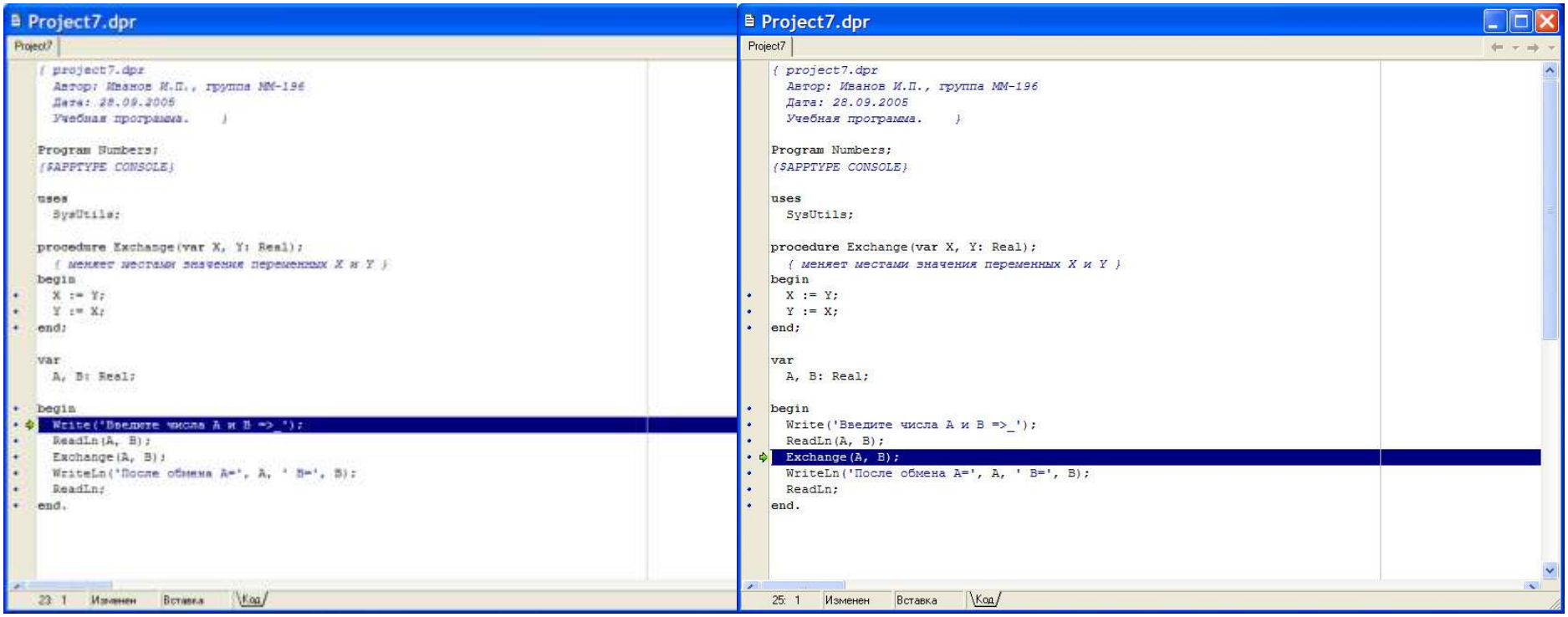


Отладчик

- *Отладчик* – программа, позволяющая отслеживать процесс выполнения программы по ее исходному тексту и просматривать текущие значения переменных.

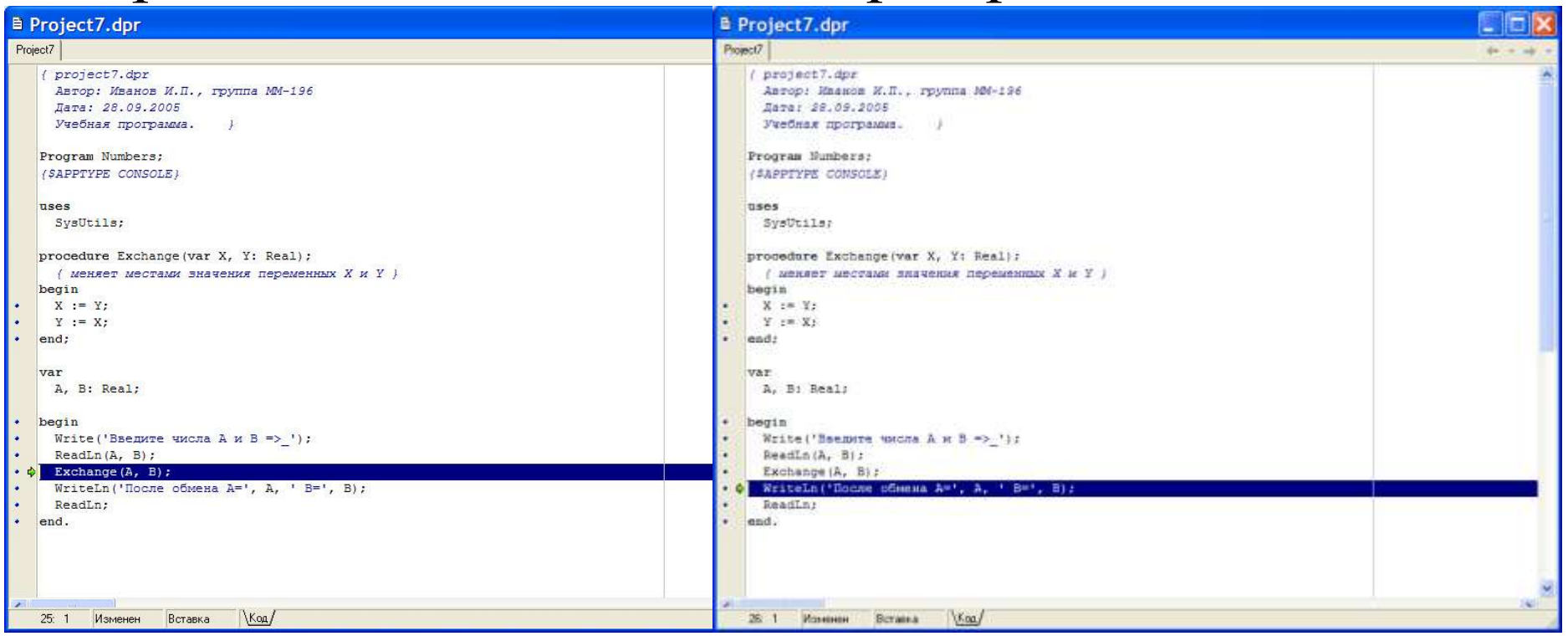
Трассировка программы

- *Трассировка* – пошаговое выполнение программы. Шагу соответствует *одна строка исходного текста* (в которой может быть более одного оператора).



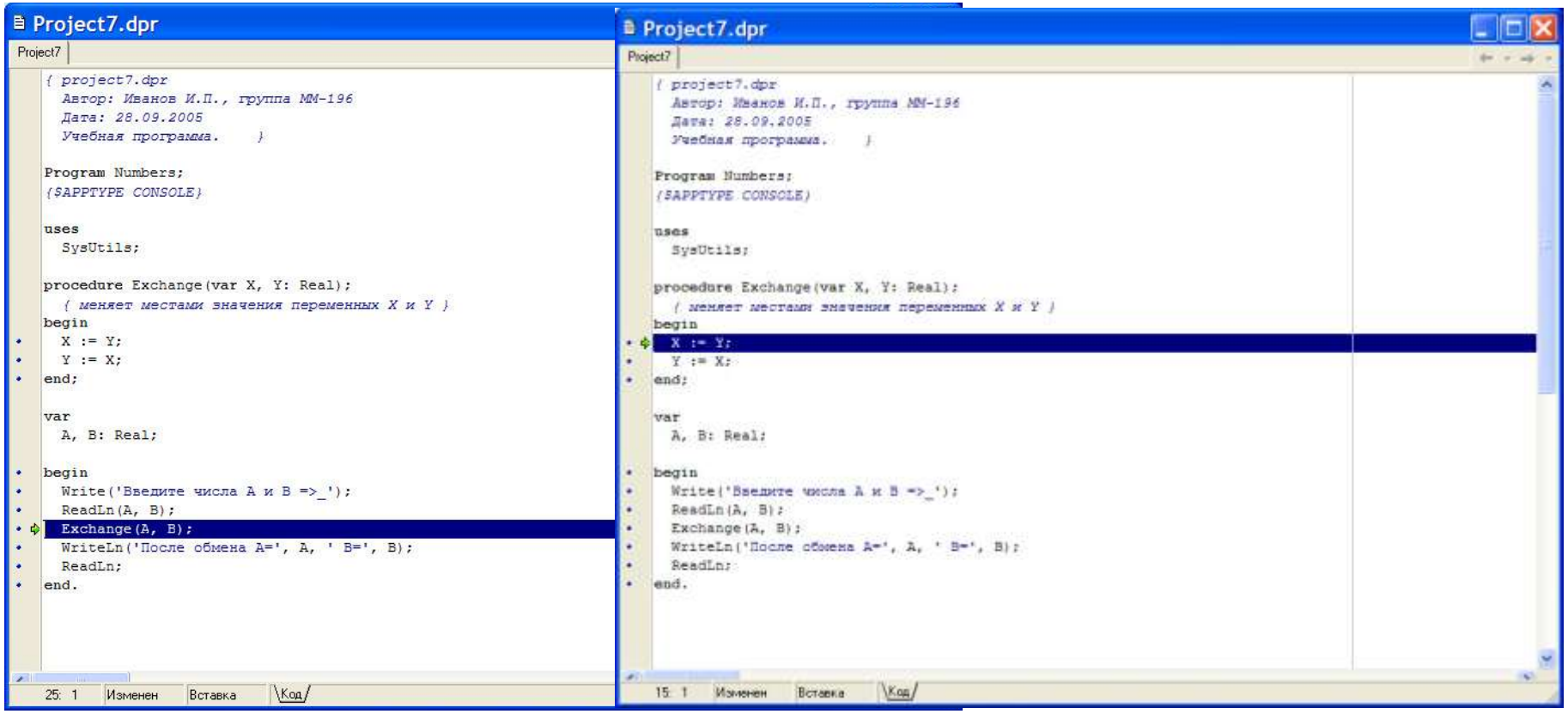
Трассировка программы

- *Режим "без трассы подпрограмм"* – пошаговое выполнение программы, при вызов подпрограммы отрабатывается как один оператор.

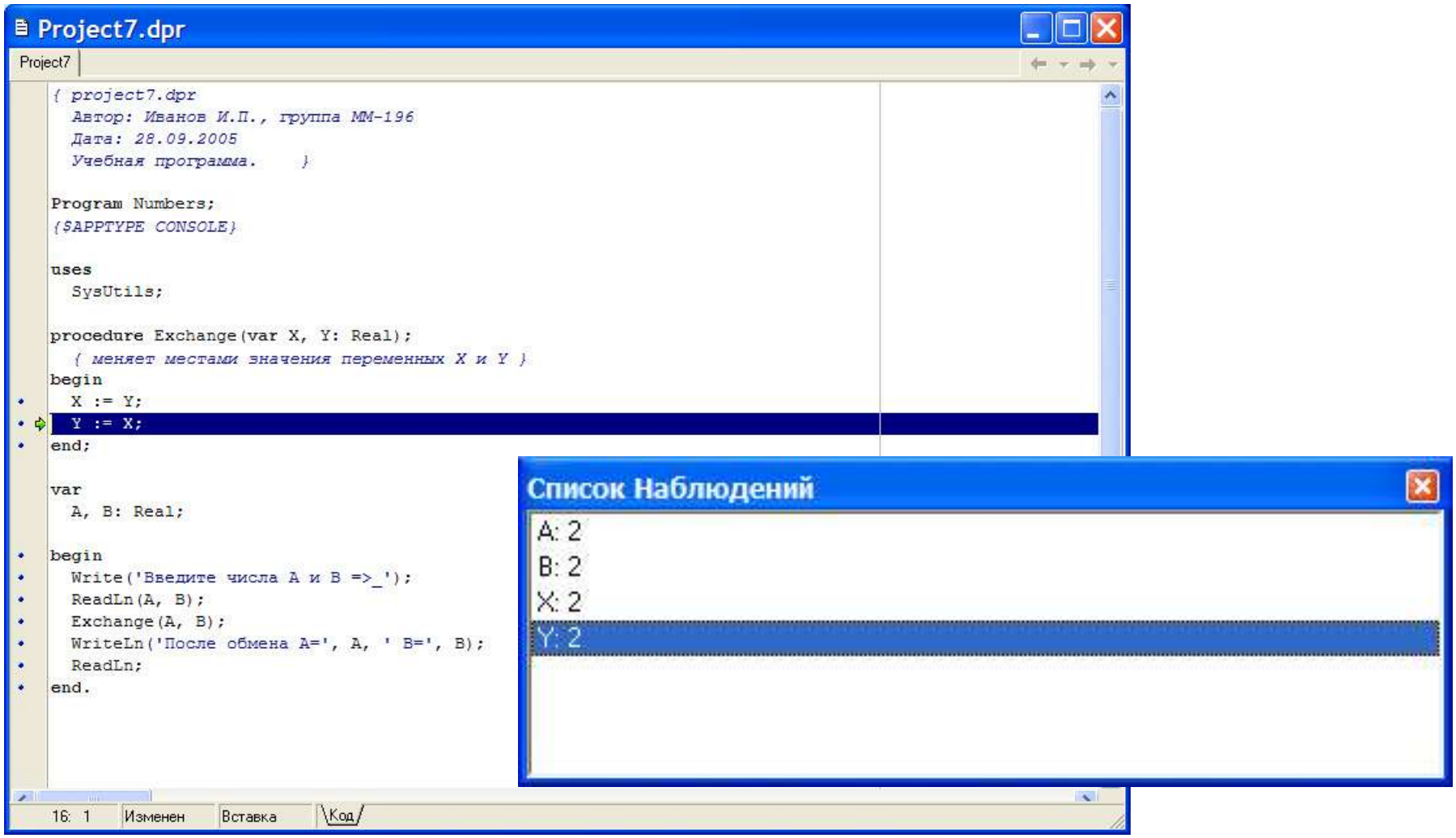


Трассировка программы

- *Режим "трасса подпрограмм"* – пошаговое выполнение программы, при котором трасса

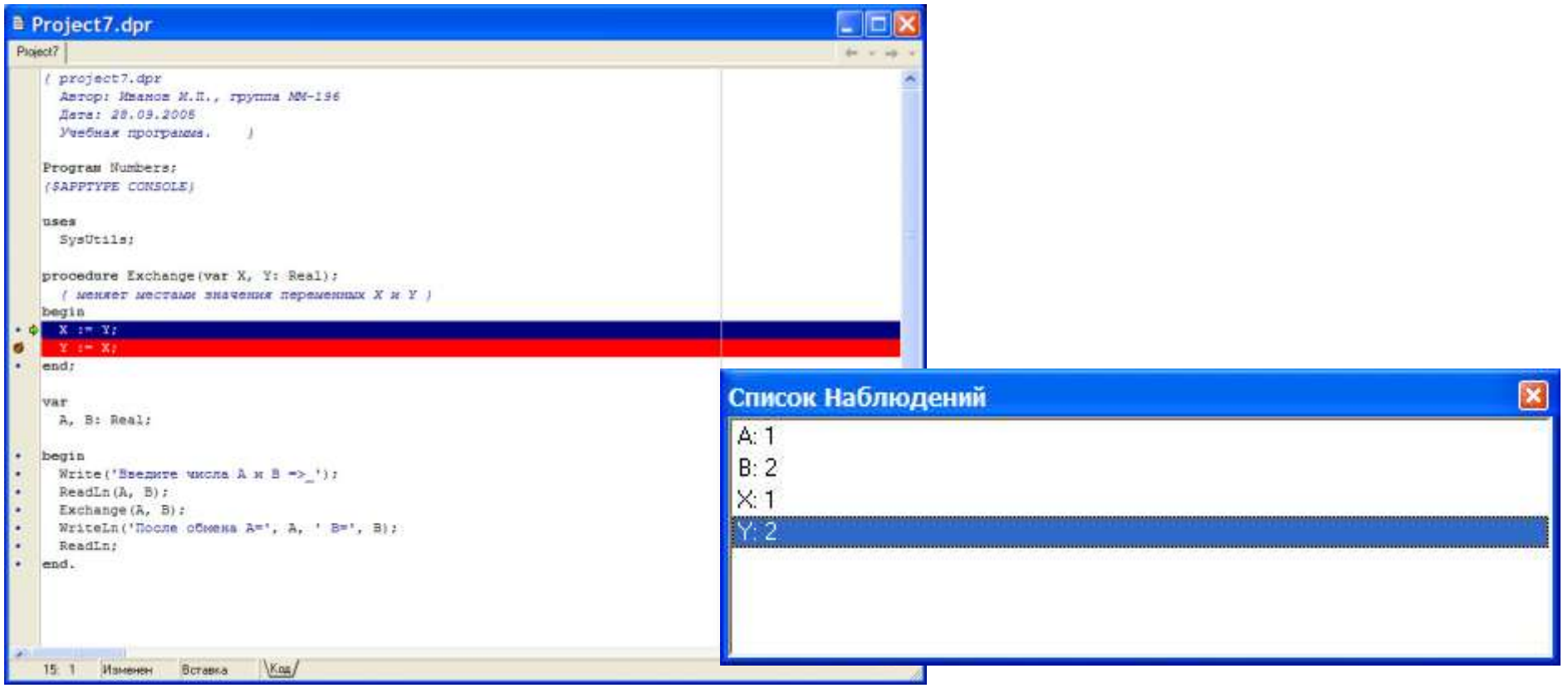


Просмотр значений переменных



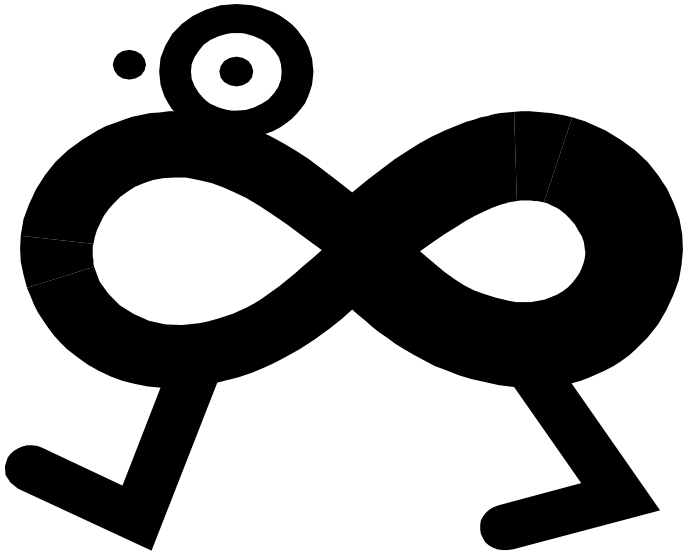
Точка останова

- *Точка останова (breakpoint)* приостанавливает выполнение программы.
  - Может быть установлена *только на выполняемом операторе* (не на комментарии и др.).
  - С точкой останова может быть связано логическое условие ее включения.

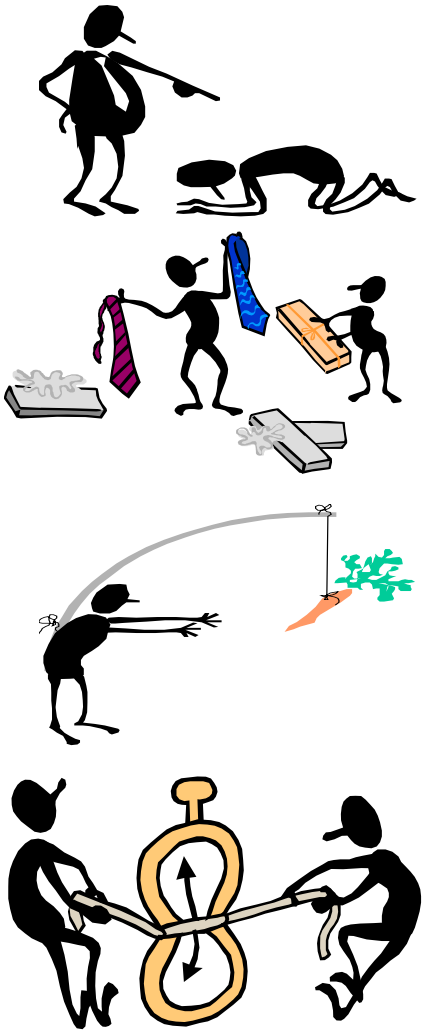


От отладки снова к тестам!

- После исправления ошибки необходимо заново выполнить тестирование на *всех* тестах.
  - При исправлении ошибки в программу можно внести новые ошибки.

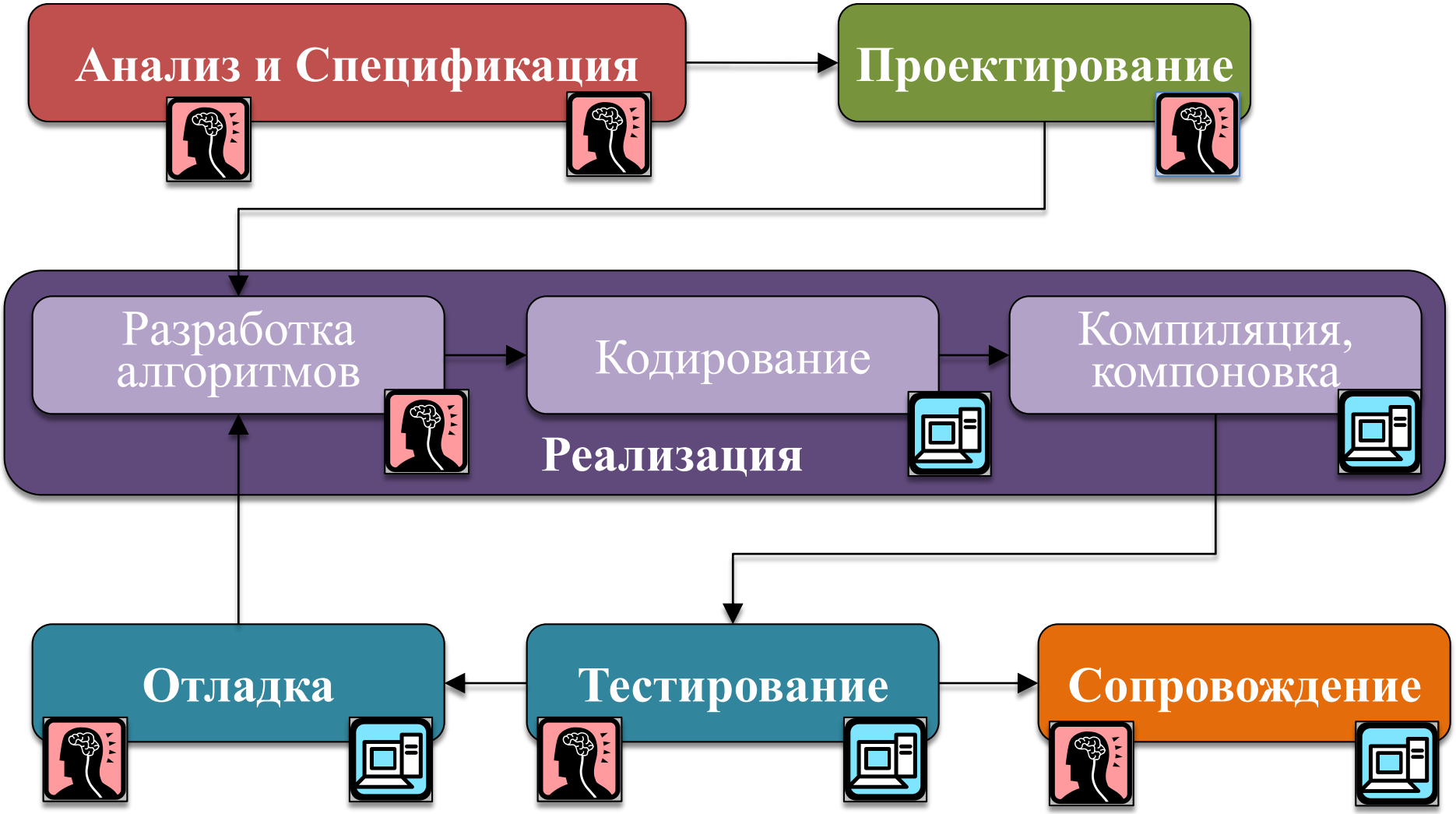


Командная разработка



- Какие роли должны быть в команде и как их распределить?
- Какие концептуальные и программные средства выбрать в качестве стандартных в команде?
- Как повысить мотивацию участников проекта?
- Что делать, если проект не укладывается в сроки?

Жизненный цикл ПО



Заключение

- Технология программирования – комплекс средств для борьбы со сложностью.
- Жизненный цикл ПО на примере
  - Анализ и спецификация
  - Проектирование
    - Модули, иерархия модулей
  - Реализация
    - Сверху вниз
  - Тестирование
    - Тестов много не бывает
  - Отладка
    - "Сухая" отладка, отладка с использованием отладчика