



ТЕХНОЛОГИЯ РАЗРАБОТКИ ПРОГРАММ

*Начинать важное дело без серьезной подготовки
есть первый признак шизофрении.*

В.М. Бехтерев

Содержание

2

- Технология программирования
- Жизненный цикл программного обеспечения
- Пример разработки программы

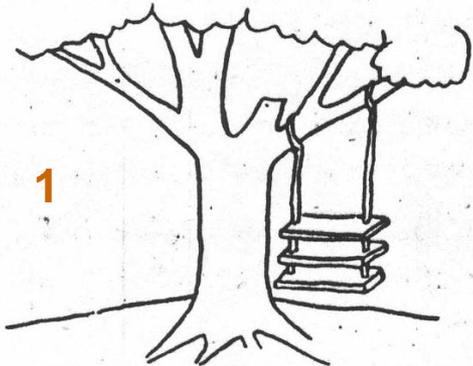
Технология программирования

3

- *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.

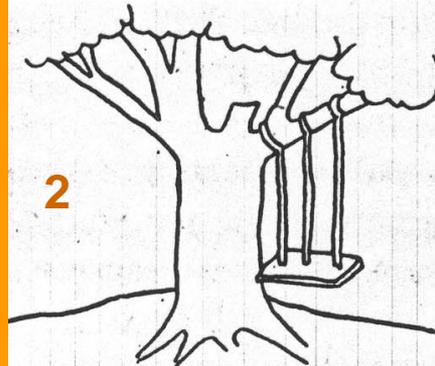
Технология программирования

4



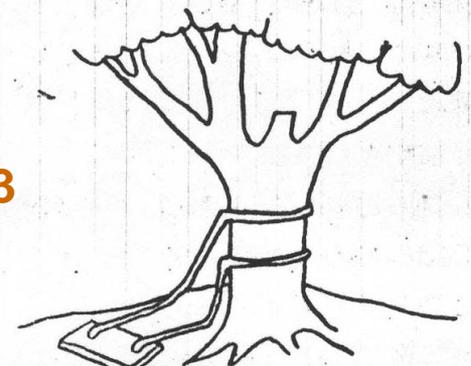
1

Как было предложено организатором разработки



2

Как было описано в техническом задании



3

Как было спроектировано ведущим системным специалистом



4

Как было реализовано программистами



5

Как было внедрено



6

Чего хотел пользователь

Как решать задачу?

5

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.

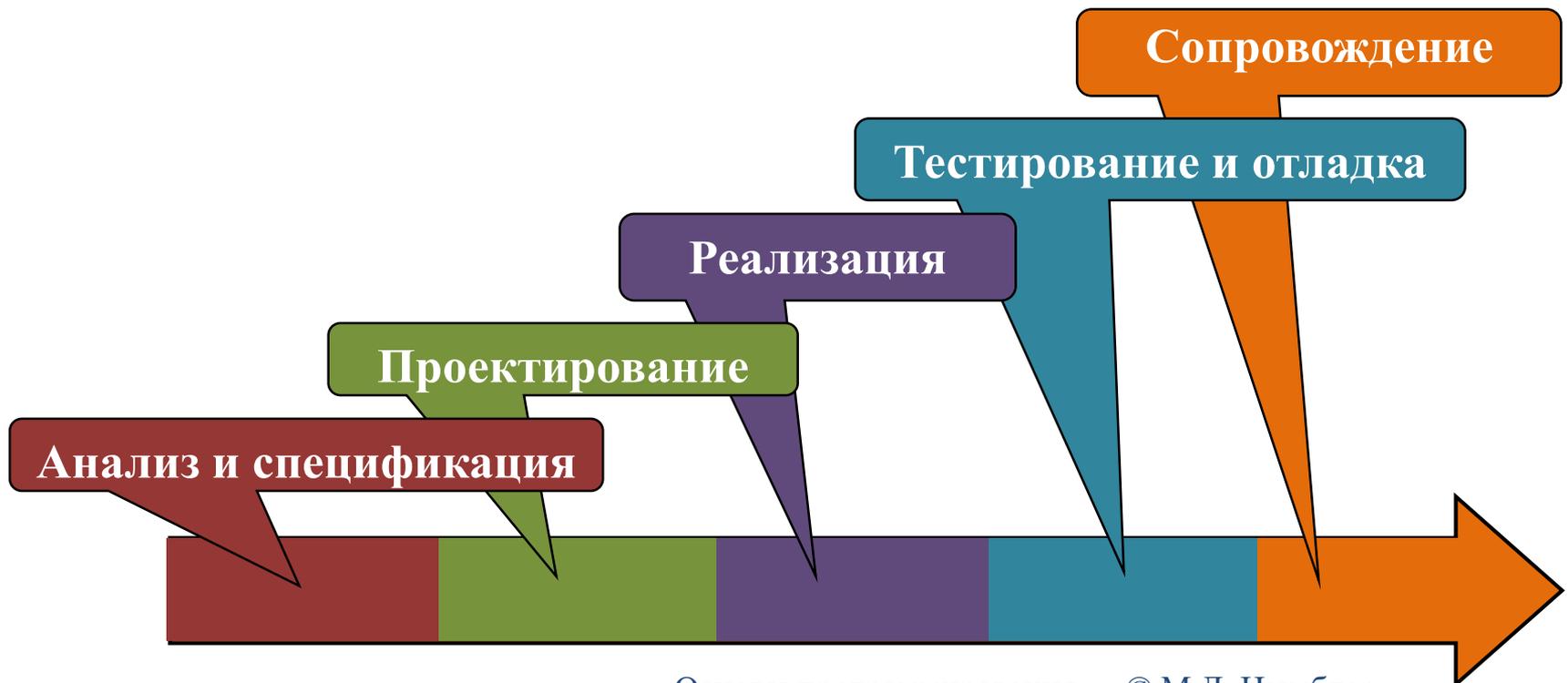


NB! Каждый из этих пунктов представляет собой задачу 😊.

Жизненный цикл ПО

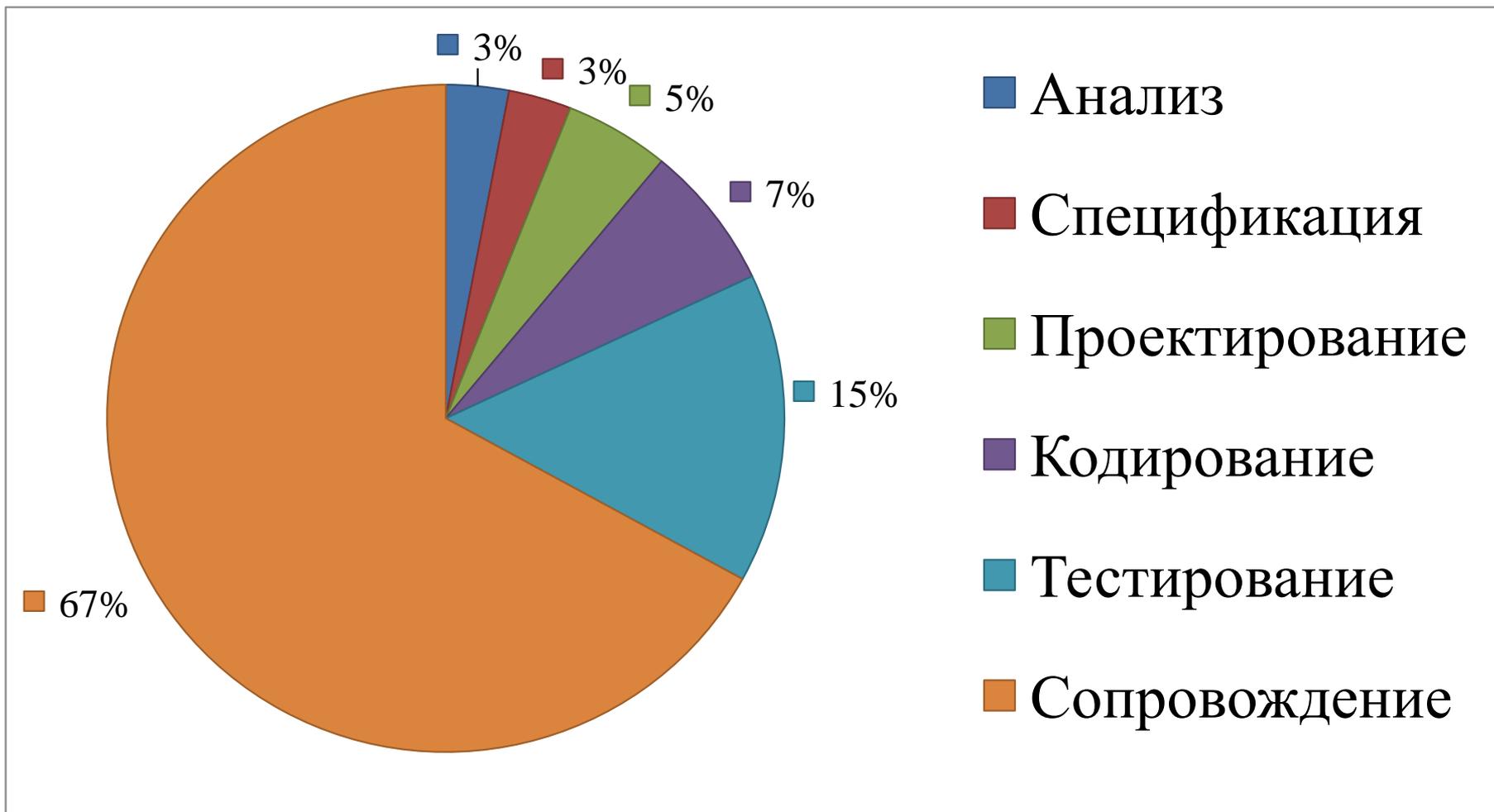
6

- *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

7



Жизненный цикл ПО

8

Анализ и Спецификация

Проектирование

Разработка алгоритмов

Кодирование

Компиляция,
компоновка

Реализация

Отладка

Тестирование

Сопровождение

Задача

9

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

Анализ требований

10

- 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

11

□ **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

12

- 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

13

□ **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

14

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

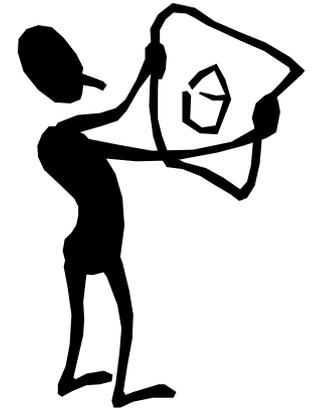
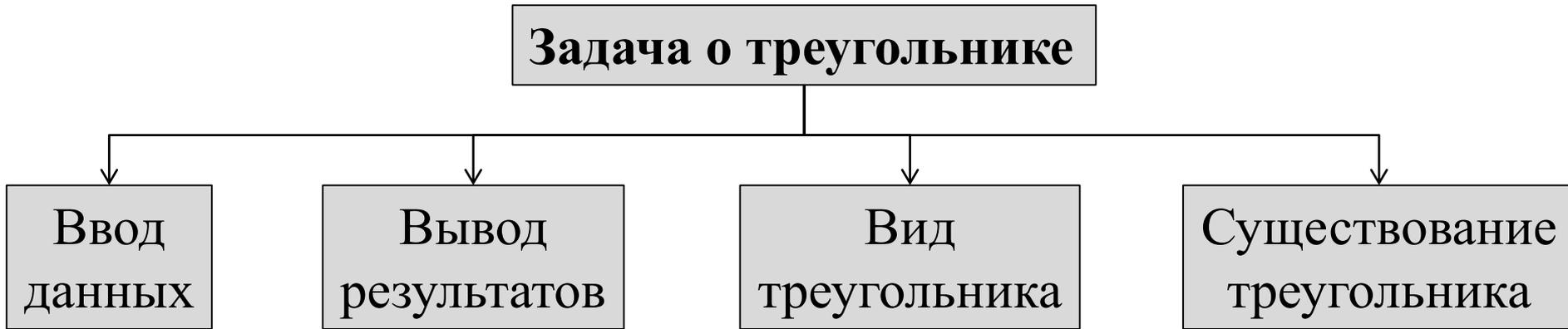


Схема модульной структуры

15

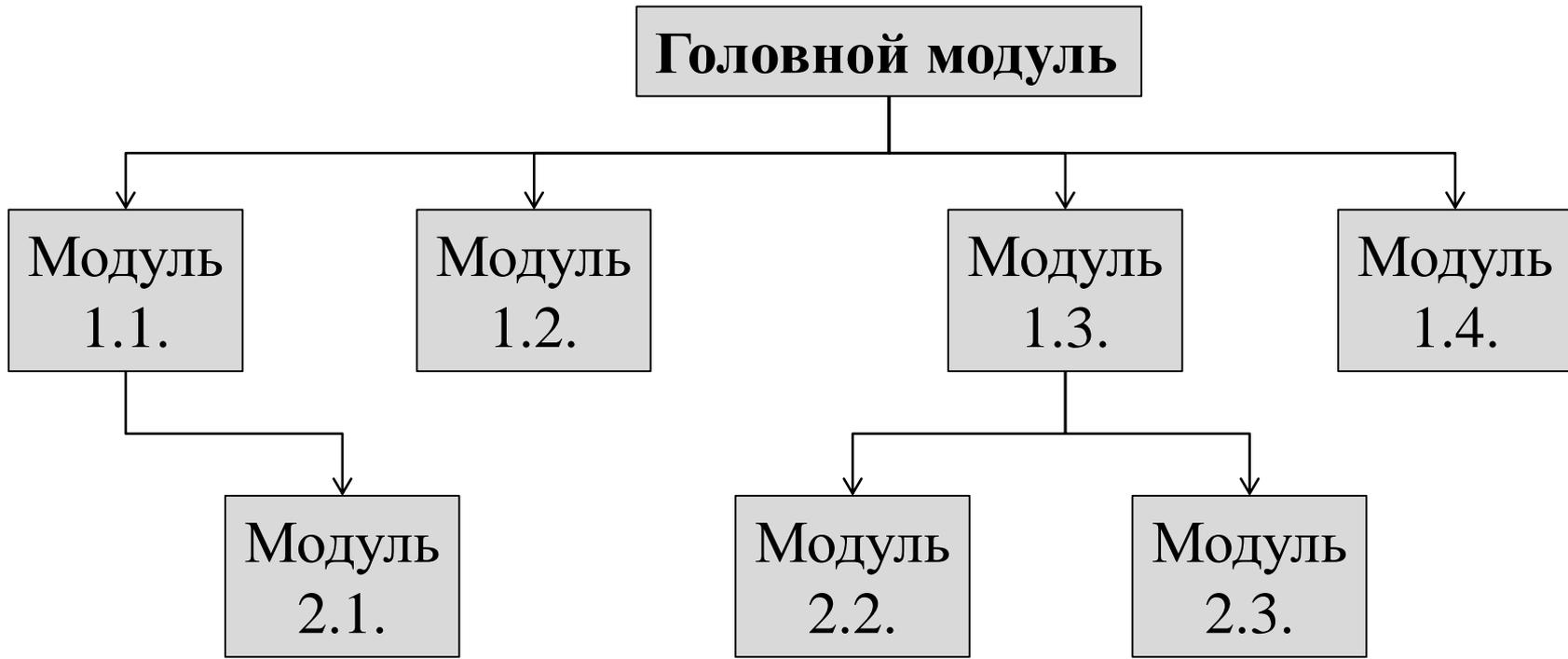


- В схеме модульной структуры стрелки показывают *связи по управлению* (главный-подчиненный), а не по времени (раньше-позже).
- Изменение реализации подчиненного модуля влечет изменение поведения главного по отношению к нему модуля.

Иерархия модулей

16

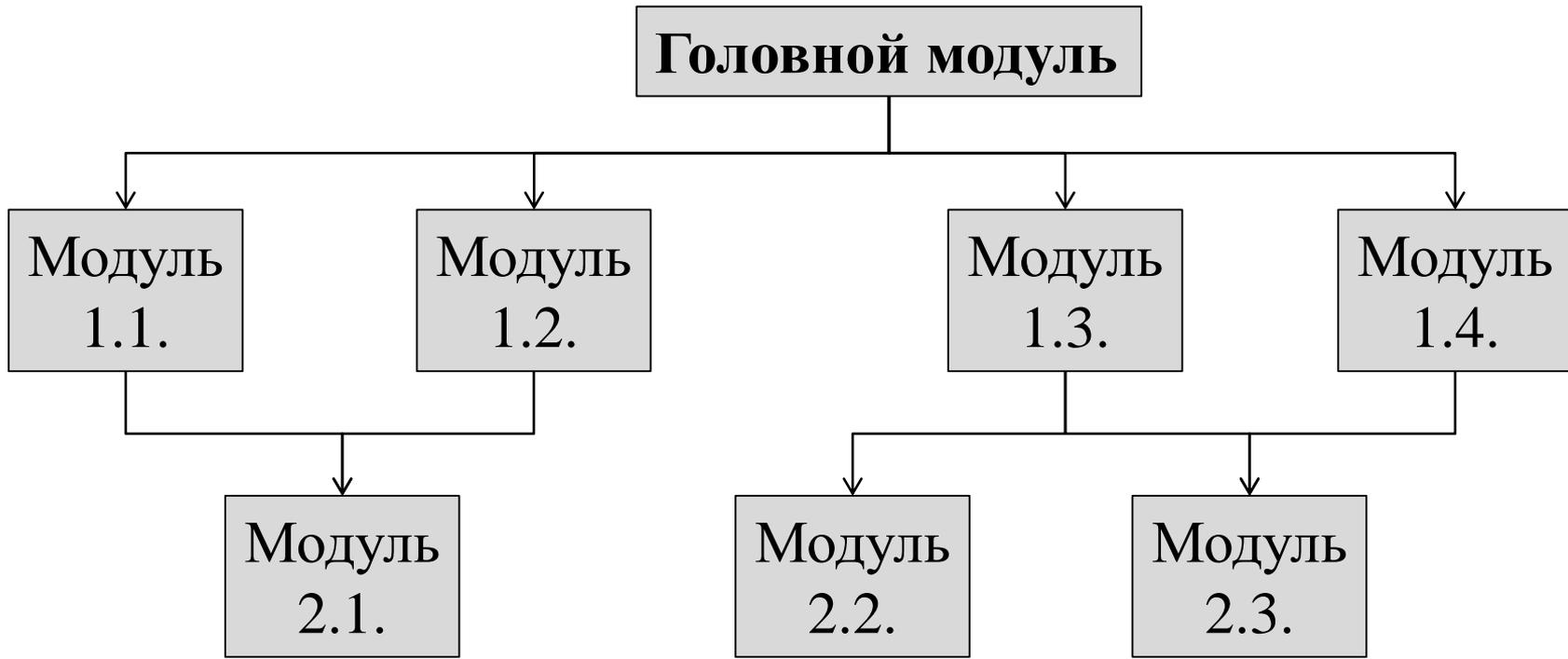
□ Сильная иерархия



Иерархия модулей

17

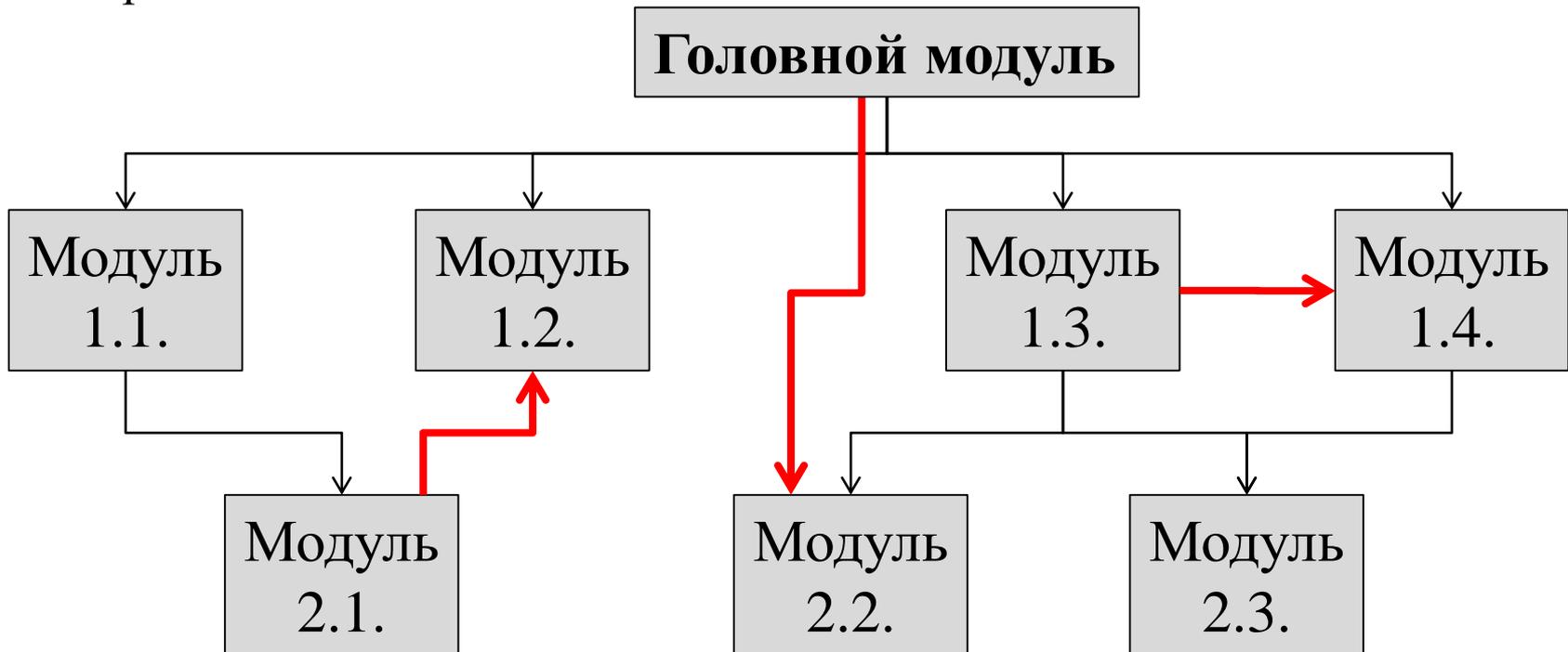
□ Ослабленная иерархия



Иерархия модулей

18

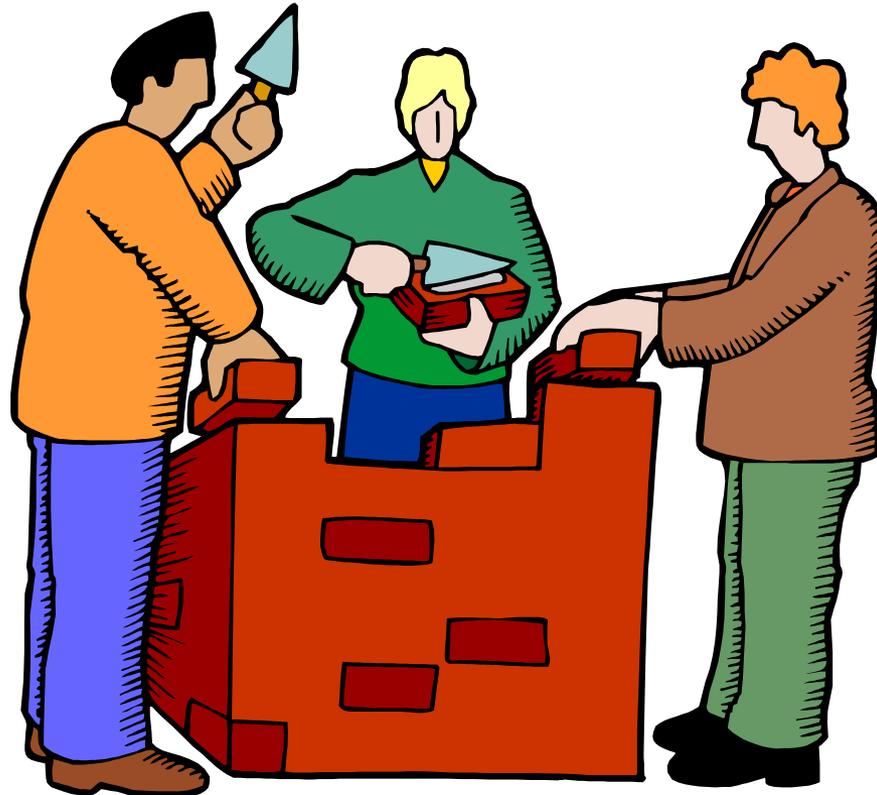
- **Недопустимые** межмодульные связи
 - ▣ снизу вверх
 - ▣ через уровень
 - ▣ горизонтальные



Проектирование: модули

19

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



Модули

20

Задача о треугольнике

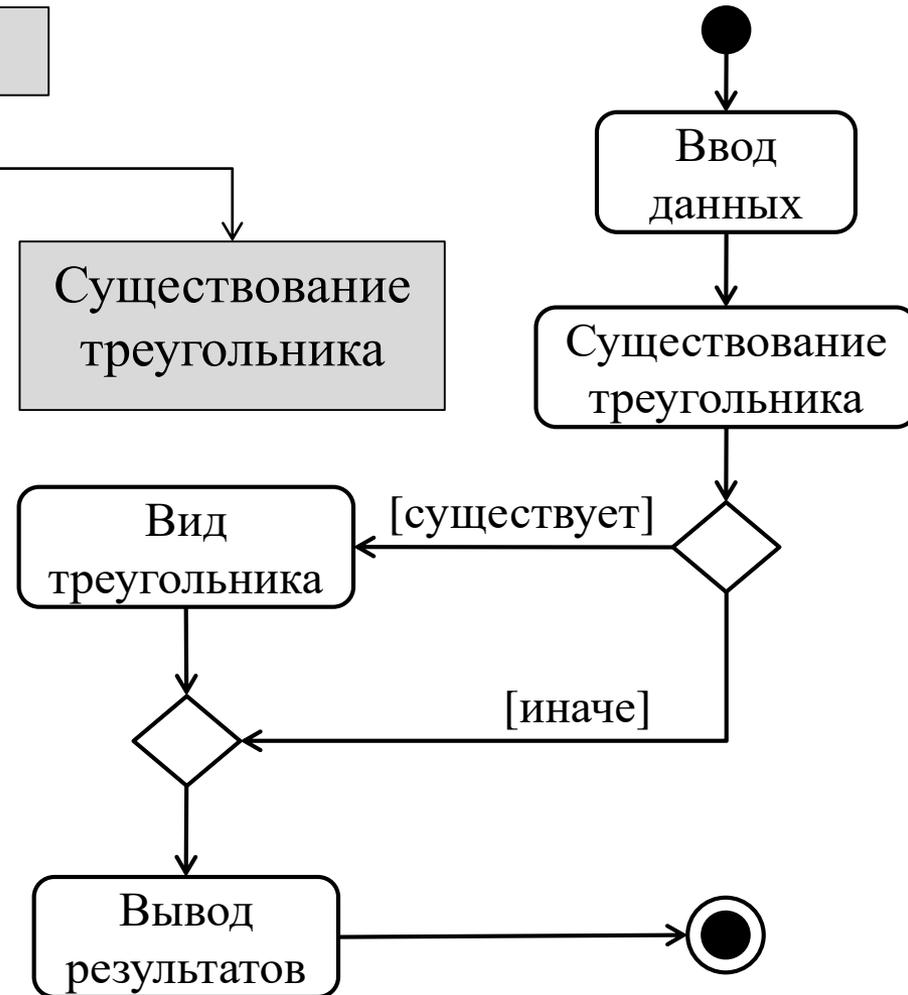
Ввод
данных

Вывод
результатов

Вид
треугольника

Существование
треугольника

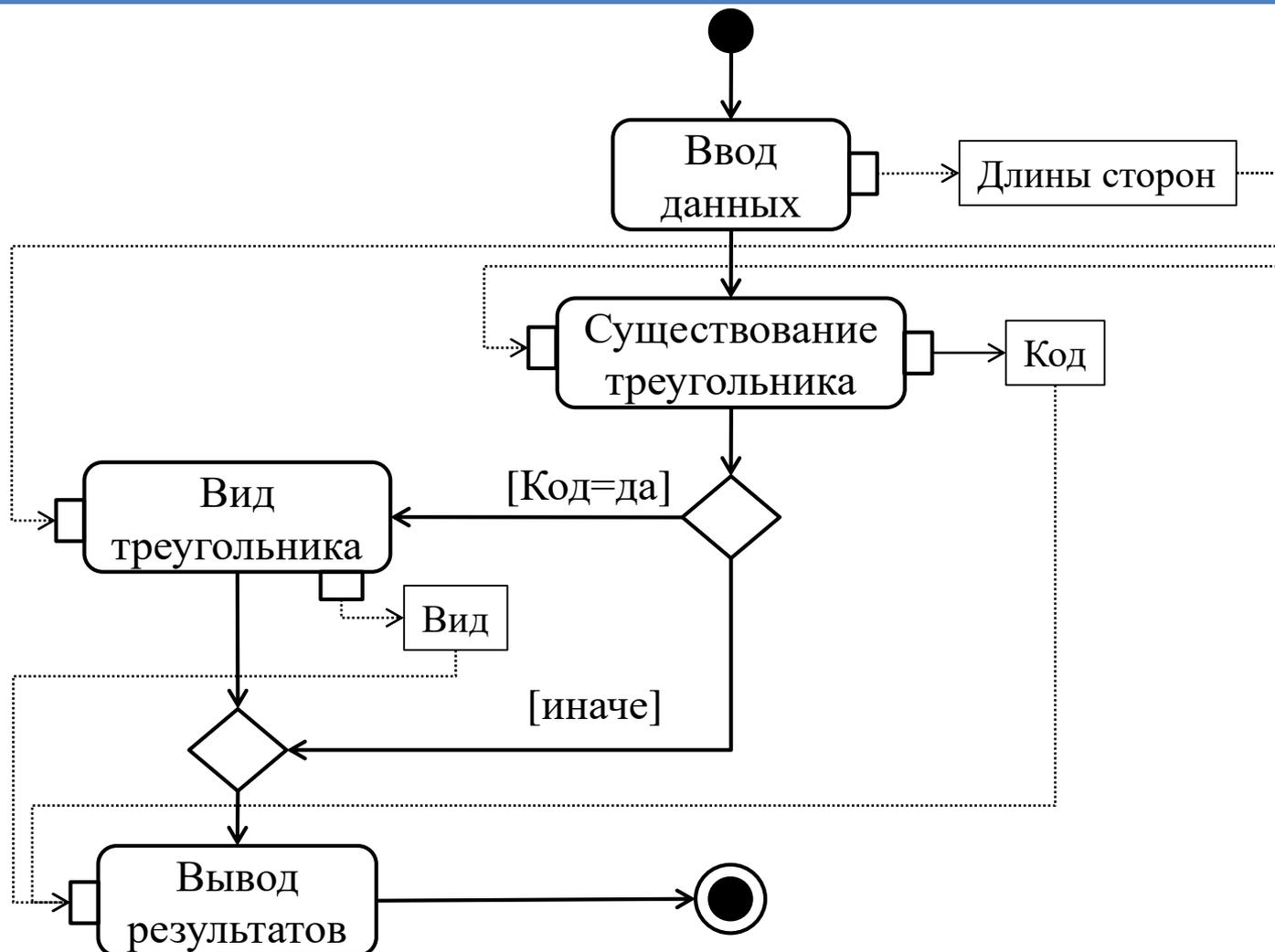
- Композиция спроектированных модулей должна обеспечивать решение задачи.



Взаимодействие модулей

21

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



Спецификации модулей

22

□ **Модуль: Ввод данных**

▣ **Назначение**

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

▣ **Входные данные**

- нет

▣ **Результаты**

- а вещ
- b вещ
- с вещ

Спецификации модулей

23

□ **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

24

□ **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

25

□ **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

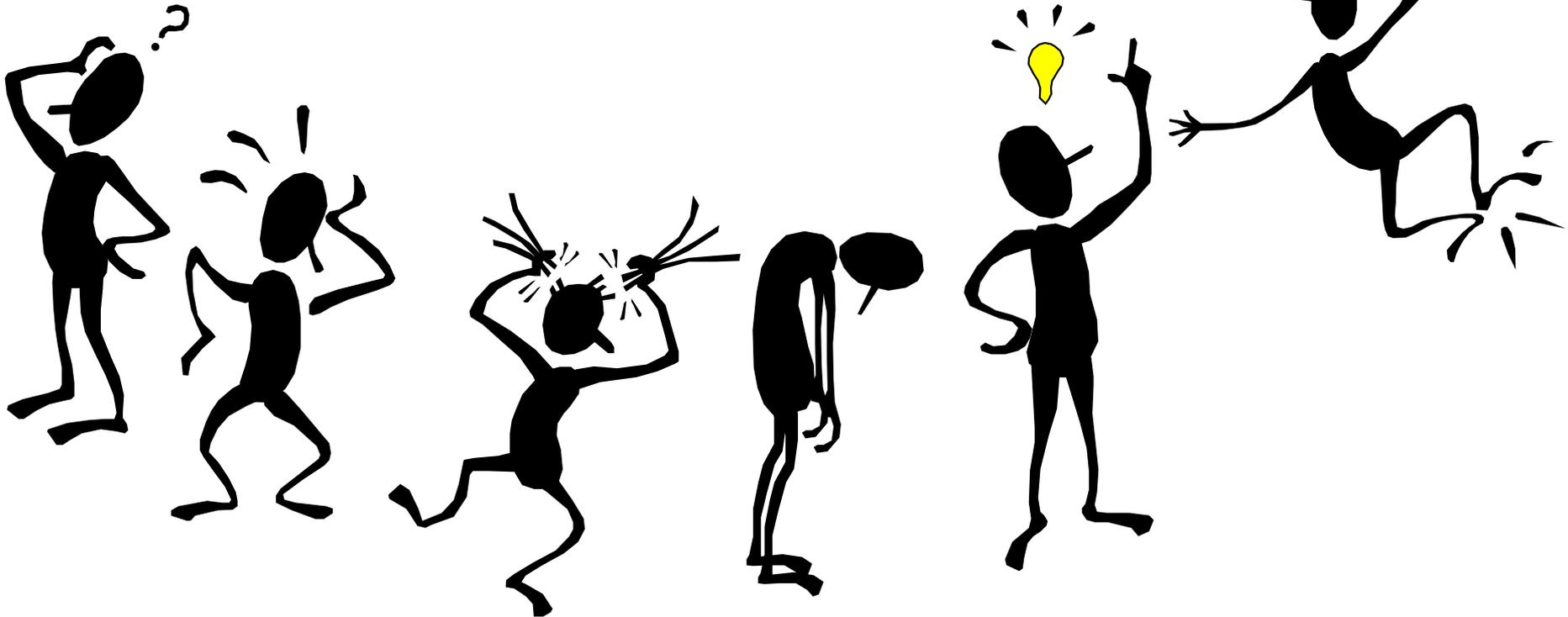
▣ **Результаты**

- kind цел
 - 3 треугольник равносторонний
 - 2 треугольник равнобедренный
 - 1 треугольник общего вида

Разработка алгоритмов

26

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



Алгоритмы

27

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

Алгоритмы

28

□ 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

29

- **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

30

Анализ и Спецификация

Проектирование

Разработка алгоритмов

Кодирование

Компиляция,
компоновка

Реализация

Отладка

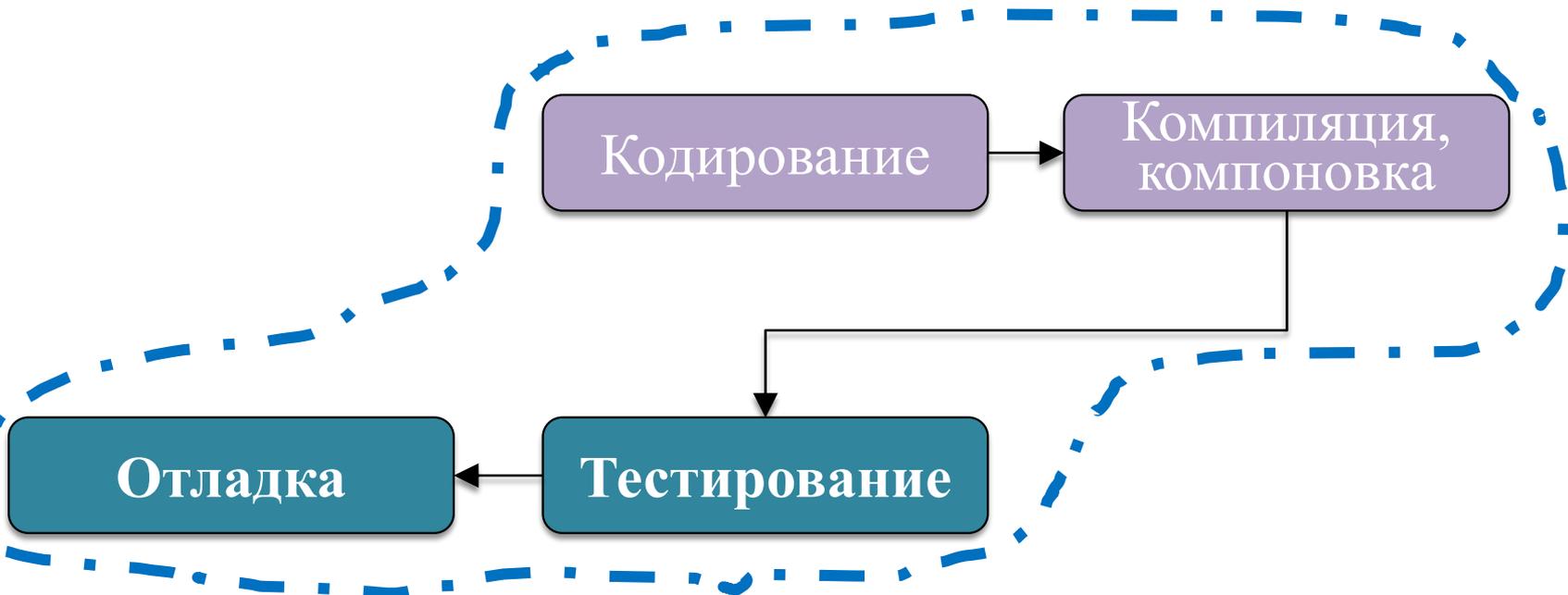
Тестирование

Сопровождение

Среда программирования

31

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



Компоненты среды

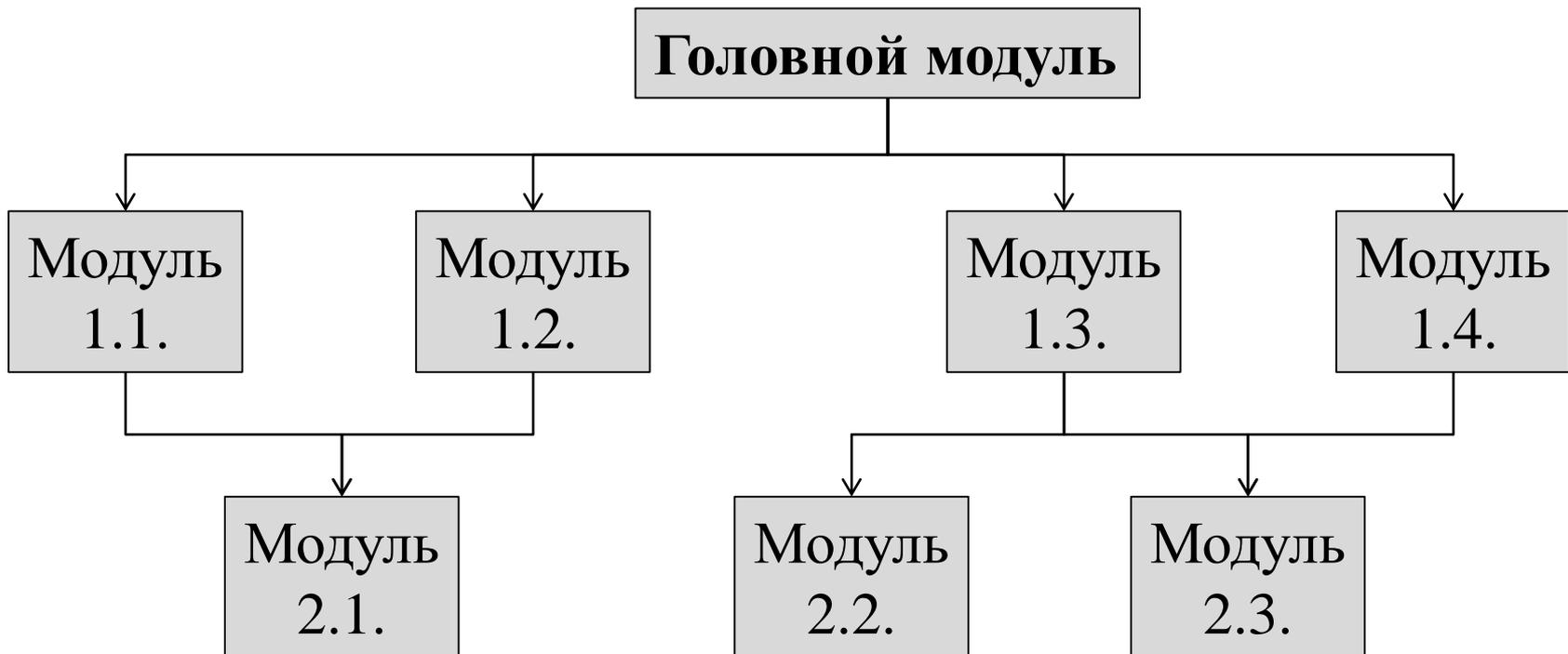
32

- *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

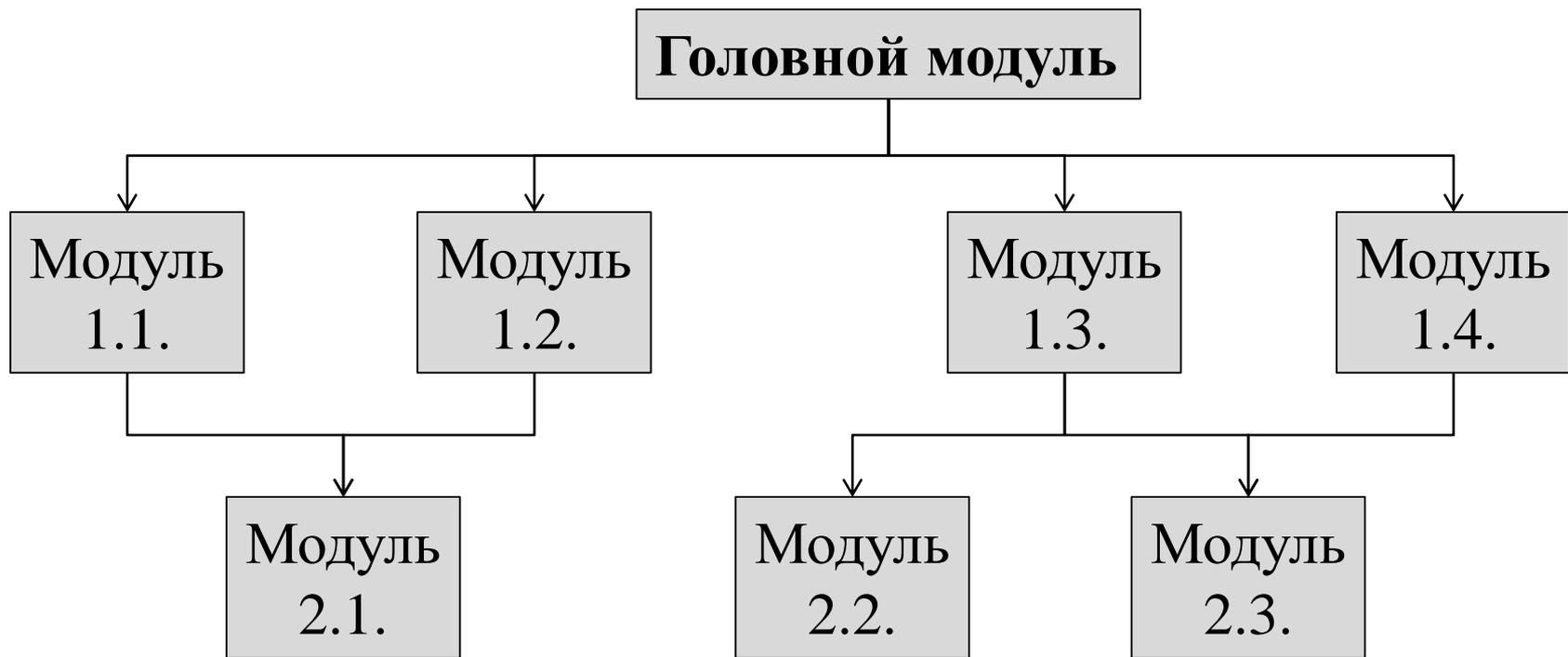
33

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



Кодирование – сверху вниз!

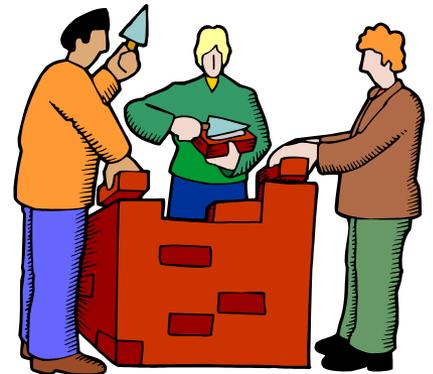
34



Кодирование: подпрограммы

35

- *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

36

```
{ TRIANGLE.PAS  
Нахождение вида треугольника.  
(с) Иванов И.И. (группа ММ-156)  
3-июля-24 }
```

Спецификация

```
Program Triangle;
```

Заголовок

```
procedure InpData (...
```

Другие модули

```
begin  
  { Пусто! }  
end.
```

Тело

Компиляция и компоновка

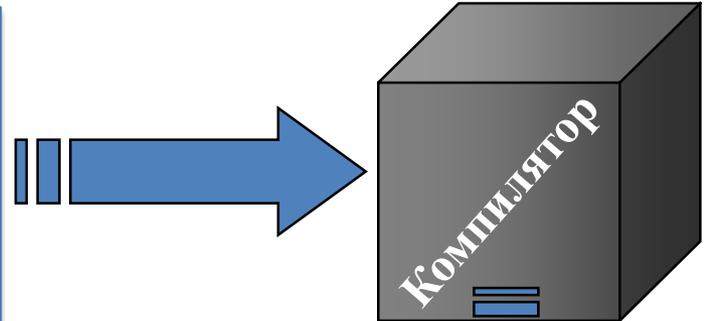
37

```
{ TRIANGLE.PAS
Нахождение вида треугольника.
(с) Иванов И.И. (ММ-156)
3-июля-24 }
```

```
Program Triangle;
```

```
begin
```

```
end.
```

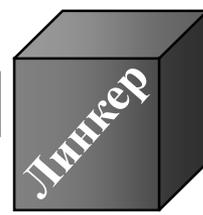


triangle.obj

```
TRIANGLE_TEXTCODE
DATADATA_BSS
BSSDGROUPSШ
("Ш Н
Ш Н Ъ __UM
_printf qИ у #FI
р {P _main ZИ
:a$ УЛьh Ъ Г-]ыб
!V-¶Oa
шTRIANGLE.PAS°
кЕ*¶И ш
```

triangle.exe

```
TRIANGLE_TEXTCODE
+X+ -!sщп| -ИБ ИГ
+t+@ O++р &!Зэ.OC+!+щ| _
6s_6q_6
ÿ_Г_0щ|d__Иб+__^]- УльVЛv
Vщ+_Л|^]-
Л^Г_ $wA_гWЛFЛN
+}A~ t&|-Gу+ÿ+Г+
ыЕ+-уе6И¶IF
```



Синтаксическая ошибка

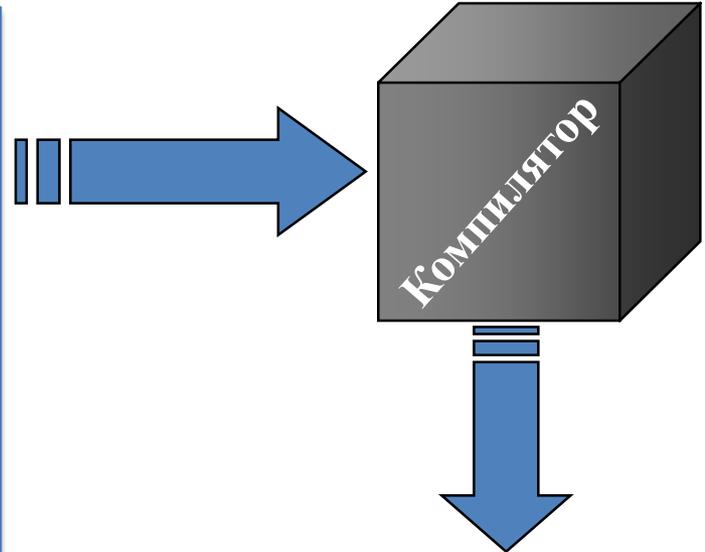
38

```
{ TRIANGLE.PAS  
Нахождение вида треугольника.  
(с) Иванов И.И. (ММ-156)  
3-июля-24 }
```

```
Program Triangle;
```

```
begin
```

```
end
```



Error: unexpected end of file

Здесь нет точки.

Нарушены синтаксические правила
языка программирования.

Кодирование модулей

39

□ 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);
```

Заголовок

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

Спецификация

```
begin  
  { Пусто! }  
end;
```

Тело

Кодирование модулей

40

□ 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

41

- **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

42

□ 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

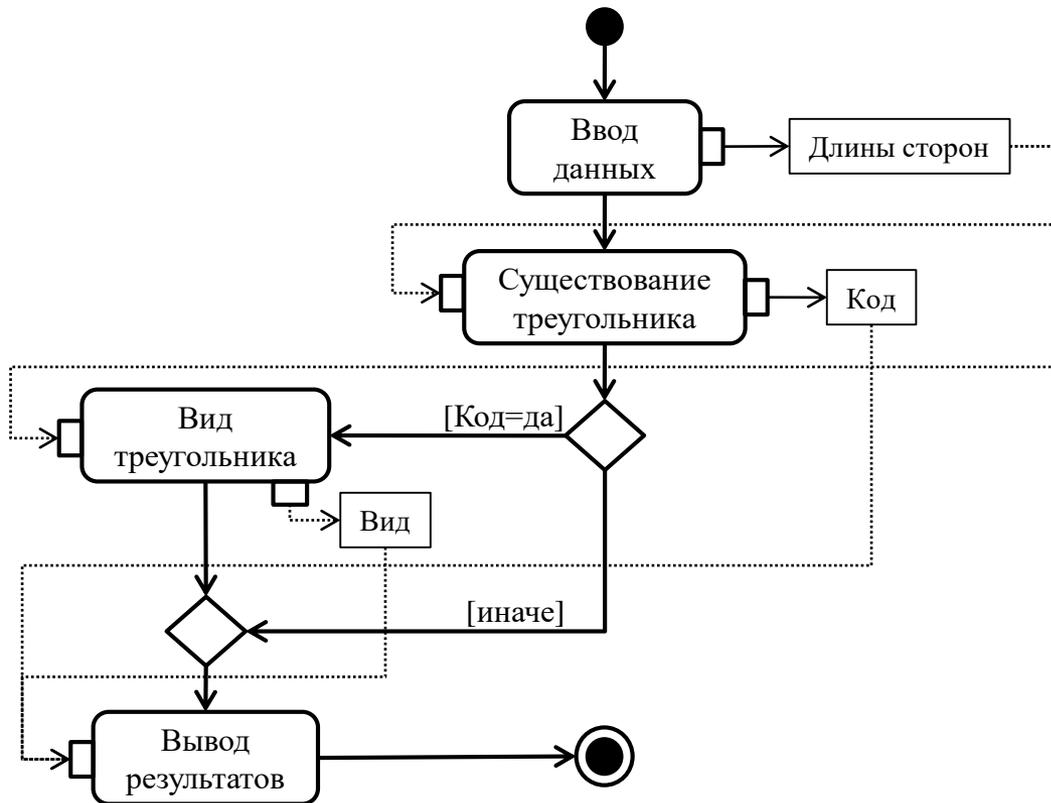
▣ Результаты

- 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

43



```
{ TRIANGLE.PAS
Нахождение вида треугольника.
(с) Иванов И.И.
3-июля-24 }
```

```
Program Triangle;
```

```
procedure InpData(...
```

```
var
```

```
a, b, c: Real;
```

```
Code: Boolean;
```

```
Kind: Integer;
```

```
begin
```

```
end.
```

Переменные

Кодирование: головной модуль

44

```
{ TRIANGLE.PAS
Нахождение вида треугольника.
(с) Иванов И.И.
3-июля-24 }
Program Triangle;

procedure InpData (...

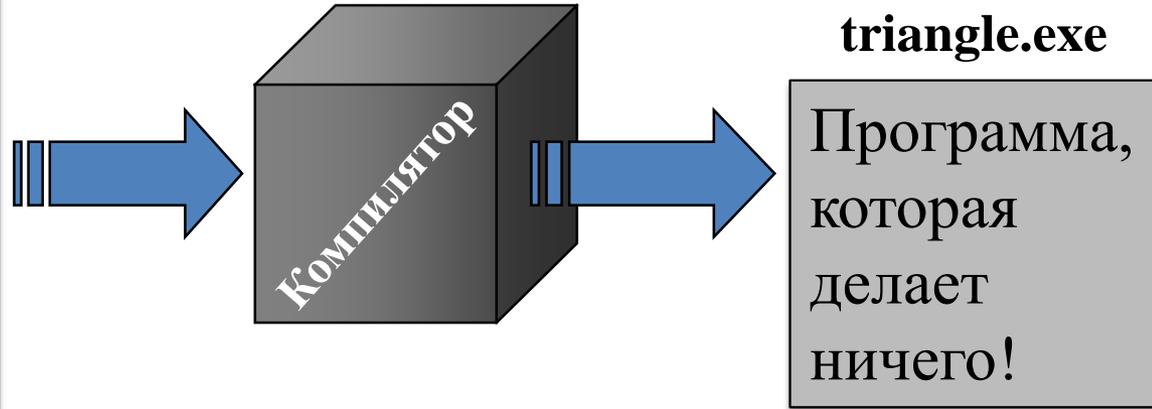
var
  a,b,c: Real;
  Code: Boolean;
  K: Integer;

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

45

```
{...}
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

46

```
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

47

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

Кодирование модулей

48

```
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

49

```
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"

50

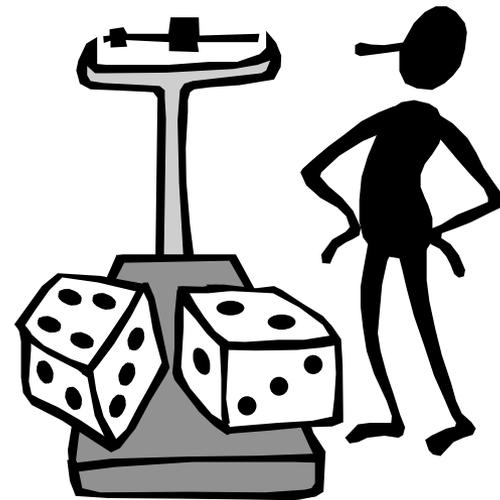
- *"Лесенка"* – отступы от левого края текста, которые показывают структурную вложенность конструкций программы.
- ▣ *"Лесенка"* не нужна компилятору:
 - можно компилировать программу, текст которой записан в 1 строку.
- ▣ *"Лесенка"* нужна программисту:
 - ясный и читаемый текст программы.



Тестирование

51

- 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

52

№	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 0BE6:0026.	✗

Ошибка времени выполнения

53

- ❑ *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.  
(с) 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!

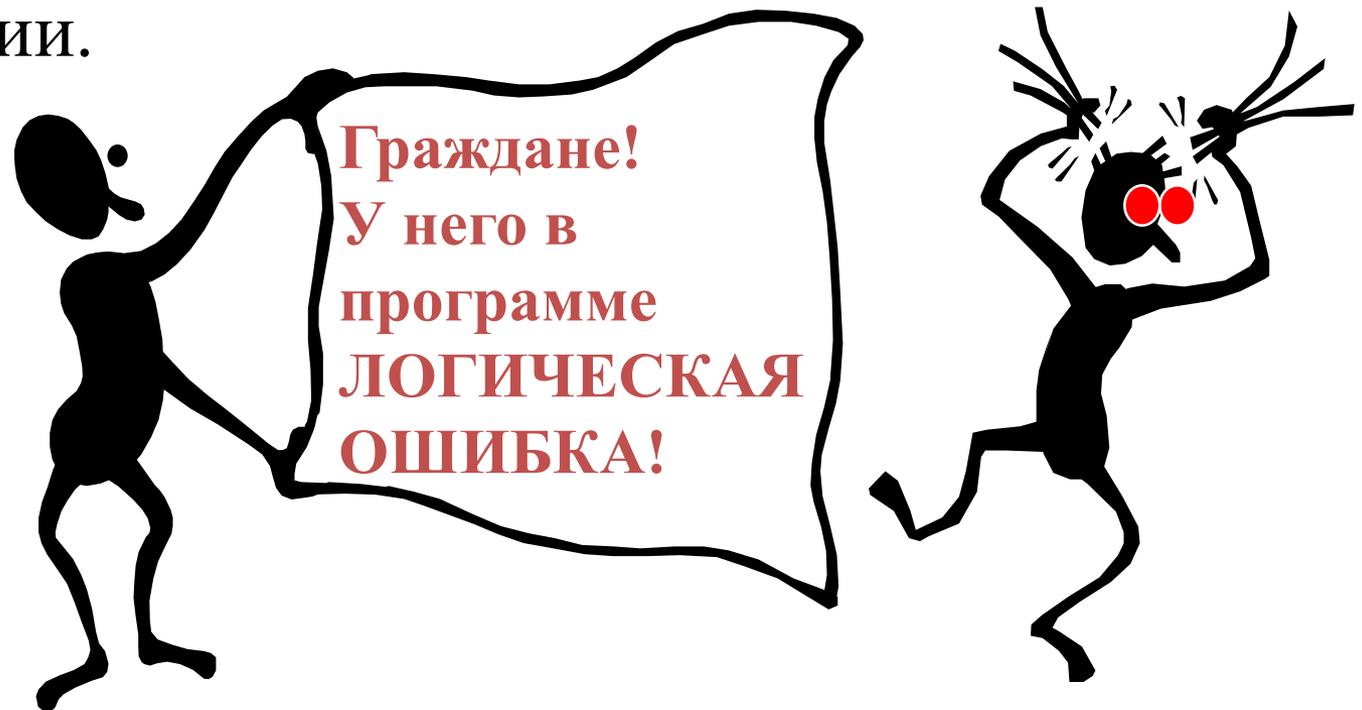
calc.exe

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

Логическая ошибка

54

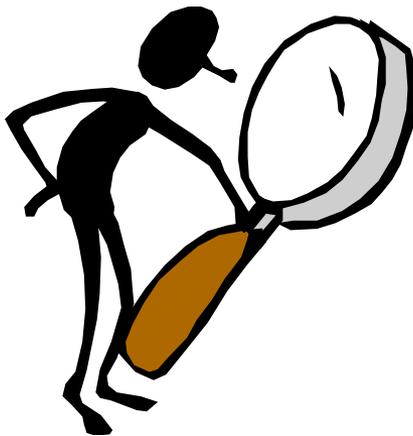
- *Логическая ошибка* – ошибка разработки алгоритма, которая заключается в том, что при выполнении программа (модуль) делает не то, что указано в спецификации.



Отладка

55

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



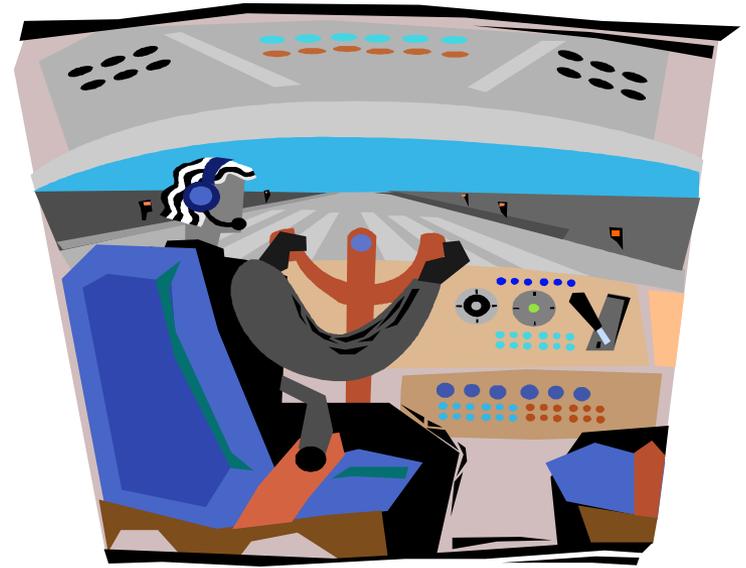
```
{Моя 333-я программа.  
(с) 2001 Иванов И. }  
Program VeryComplex;  
uses MyUnit;  
procedure CalcXY(a,b: Integer;  
  var X, Y: Real);  
var i,,j: Integer;  
begin  
  for i:=1 to Max
```



Отладчик

56

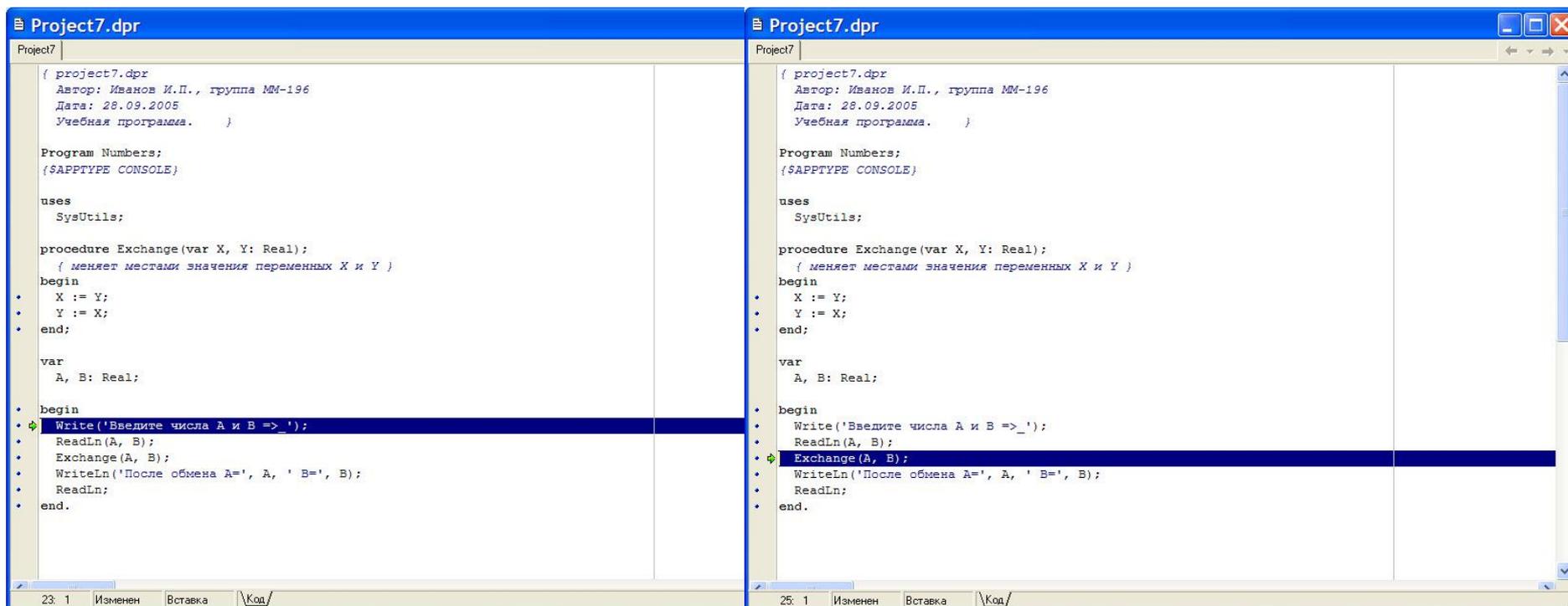
- *Отладчик* – программа, позволяющая отслеживать процесс выполнения программы по ее исходному тексту и просматривать текущие значения переменных.



Трассировка программы

57

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



```
{ project7.dpr
Автор: Иванов И.П., группа ММ-196
Дата: 28.09.2005
Учебная программа. }

Program Numbers;
{$APPTYPE CONSOLE}

uses
  SysUtils;

procedure Exchange(var X, Y: Real);
{ меняет местами значения переменных X и Y }
begin
  X := Y;
  Y := X;
end;

var
  A, B: Real;

begin
  Write('Введите числа A и B => ');
  ReadLn(A, B);
  Exchange(A, B);
  WriteLn('После обмена A=', A, ' B=', B);
  ReadLn;
end.
```

Трассировка программы

58

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

```
{ project7.dpr
Автор: Иванов И.П., группа MM-196
Дата: 28.09.2005
Учебная программа.  }

Program Numbers;
{$APPTYPE CONSOLE}

uses
  SysUtils;

procedure Exchange(var X, Y: Real);
  { меняет местами значения переменных X и Y }
begin
  X := Y;
  Y := X;
end;

var
  A, B: Real;

begin
  Write('Введите числа A и B => ');
  ReadLn(A, B);
  Exchange(A, B);
  WriteLn('После обмена A=', A, ' B=', B);
  ReadLn;
end.
```

Трассировка программы

59

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

The image shows two side-by-side screenshots of the Delphi IDE, illustrating the 'Breakpoints' mode for tracing subprograms. Both windows display the same Pascal code for 'Project7.dpr'. The code includes a procedure 'Exchange' that swaps the values of two real variables, X and Y. In the left screenshot, a blue horizontal bar highlights the line 'Exchange(A, B);' in the main procedure's call list, indicating that a breakpoint is set at this point. In the right screenshot, the same line is highlighted, but the IDE's status bar at the bottom shows '15: 1' instead of '25: 1', suggesting the program has stepped through the first 15 lines of code.

```
{ project7.dpr
Автор: Иванов И.П., группа ММ-196
Дата: 28.09.2005
Учебная программа. }

Program Numbers;
{$APPTYPE CONSOLE}

uses
  SysUtils;

procedure Exchange(var X, Y: Real);
{ меняет местами значения переменных X и Y }
begin
  X := Y;
  Y := X;
end;

var
  A, B: Real;

begin
  Write('Введите числа A и B => ');
  ReadLn(A, B);
  Exchange(A, B);
  WriteLn('После обмена A=', A, ' B=', B);
  ReadLn;
end.
```

Просмотр значений переменных

60

The screenshot displays the Delphi IDE interface. The main window, titled "Project7.dpr", shows the source code of a Pascal program. The code includes a header block with author and date information, a program declaration, and a procedure named "Exchange" that swaps the values of two real variables, X and Y. The main program body prompts the user to enter two numbers, A and B, and then calls the "Exchange" procedure. A mouse cursor is positioned over the line "Y := X;" in the procedure's body.

In the bottom right corner, a "Список Наблюдений" (Watch List) window is open. It contains a list of variables being monitored: A: 2, B: 2, X: 2, and Y: 2. The entry "Y: 2" is currently selected and highlighted with a blue background.

```
{ project7.dpr
Автор: Иванов И.П., группа ММ-196
Дата: 28.09.2005
Учебная программа. }

Program Numbers;
{$APPTYPE CONSOLE}

uses
  SysUtils;

procedure Exchange(var X, Y: Real);
  { меняет местами значения переменных X и Y }
begin
  X := Y;
  Y := X;
end;

var
  A, B: Real;

begin
  Write('Введите числа A и B => ');
  ReadLn(A, B);
  Exchange(A, B);
  WriteLn('После обмена A=', A, ' B=', B);
  ReadLn;
end.
```

Список Наблюдений

- A: 2
- B: 2
- X: 2
- Y: 2

16. 1 Изменен Вставка \Код/

Точка останова

61

- 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.

```
{ project7.dpr
Автор: Иванов И.П., группа MM-196
Дата: 28.09.2005
Учебная программа. }

Program Numbers;
{$APPTYPE CONSOLE}

uses
  SysUtils;

procedure Exchange(var X, Y: Real);
  { меняет местами значения переменных X и Y }
begin
  X := Y;
  Y := X;
end;

var
  A, B: Real;

begin
  Write('Введите числа A и B => ');
  ReadLn(A, B);
  Exchange(A, B);
  WriteLn('После обмена A=', A, ' B=', B);
  ReadLn;
end.
```

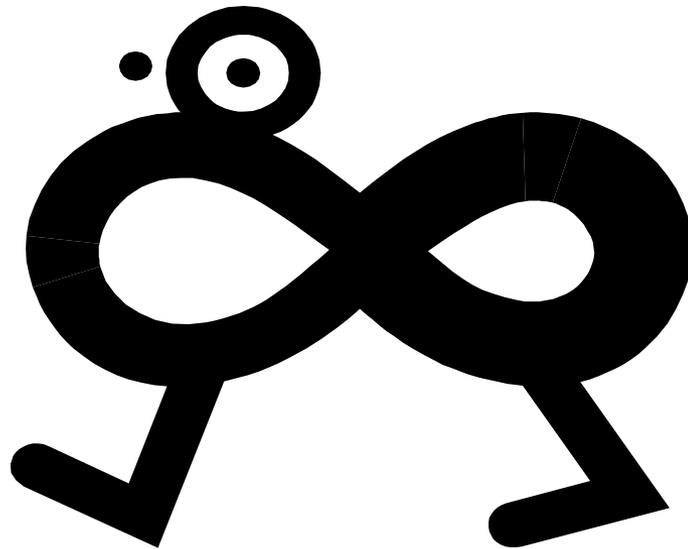
Список Наблюдений

A: 1
B: 2
X: 1
Y: 2

От отладки снова к тестам!

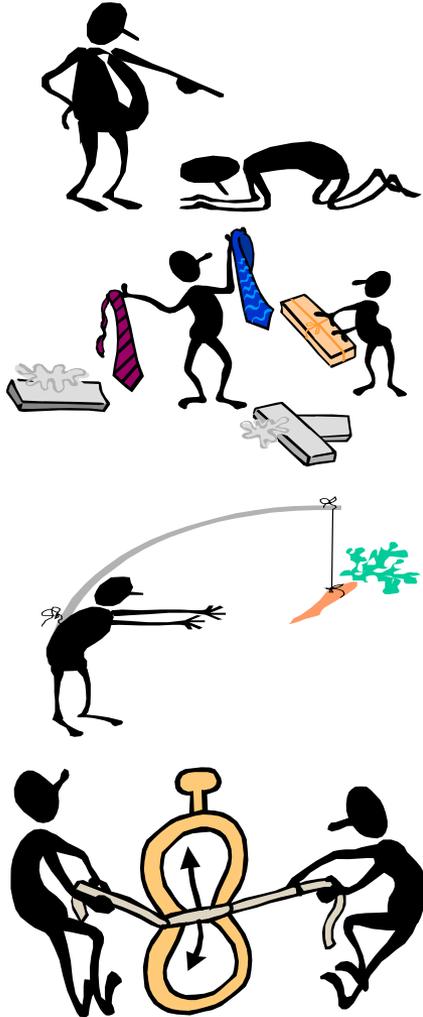
62

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



Командная разработка

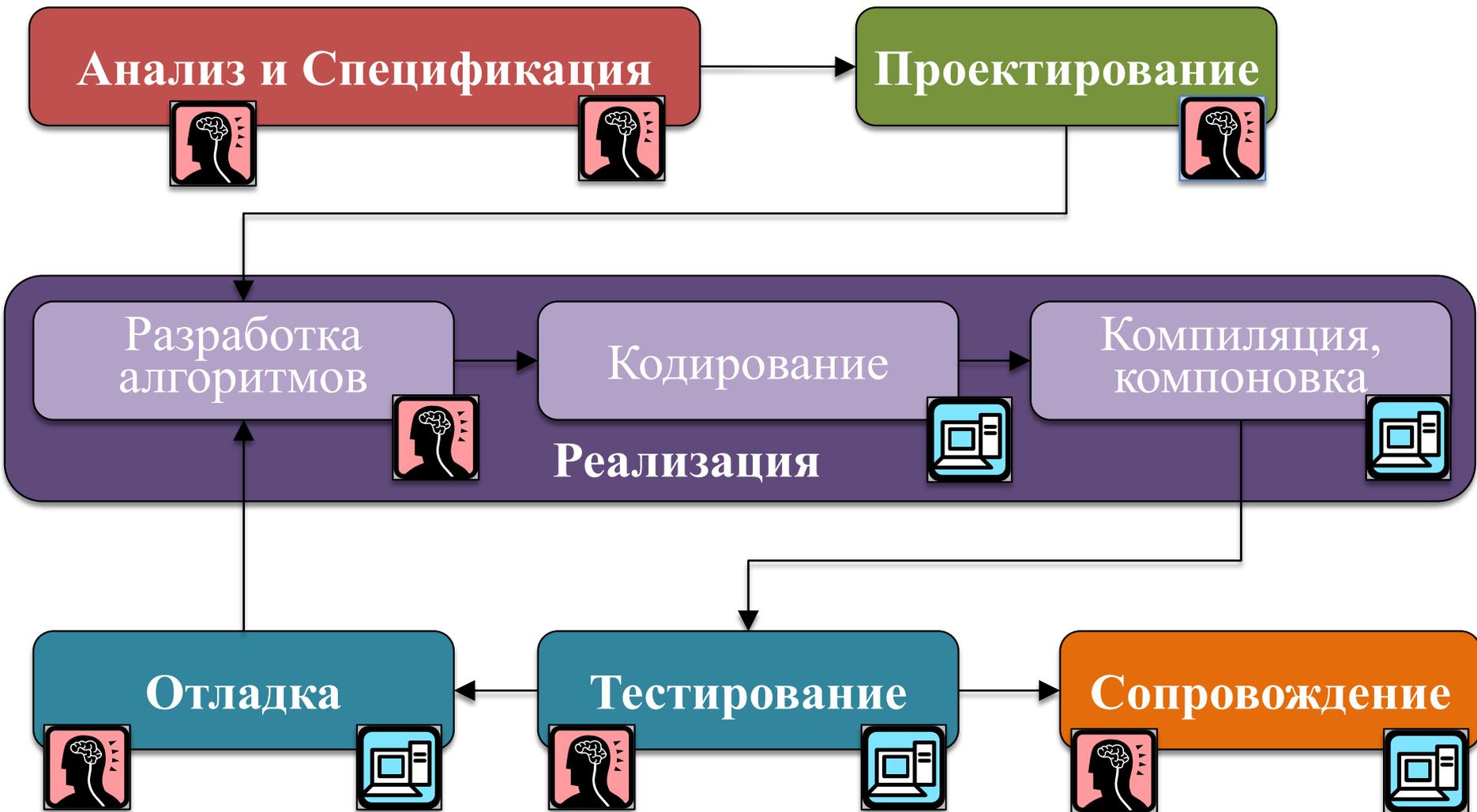
63



- 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

64



Заключение

65

- 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