Общая специфика процесса разработки программного обеспечения

От предметной области к ПО

Разработка больших программ – многоступенчатый процесс, в ходе которого осуществляются как ручные трансформации неформальных моделей решаемой задачи в формализованные представления, так и их последующая автоматическая трансляция с использованием различных систем программирования.

Предметная область
Архитектура ВС
Инструменты для разработки ПО

Примеры прикладных задач и их моделей

Разработка трансляторов

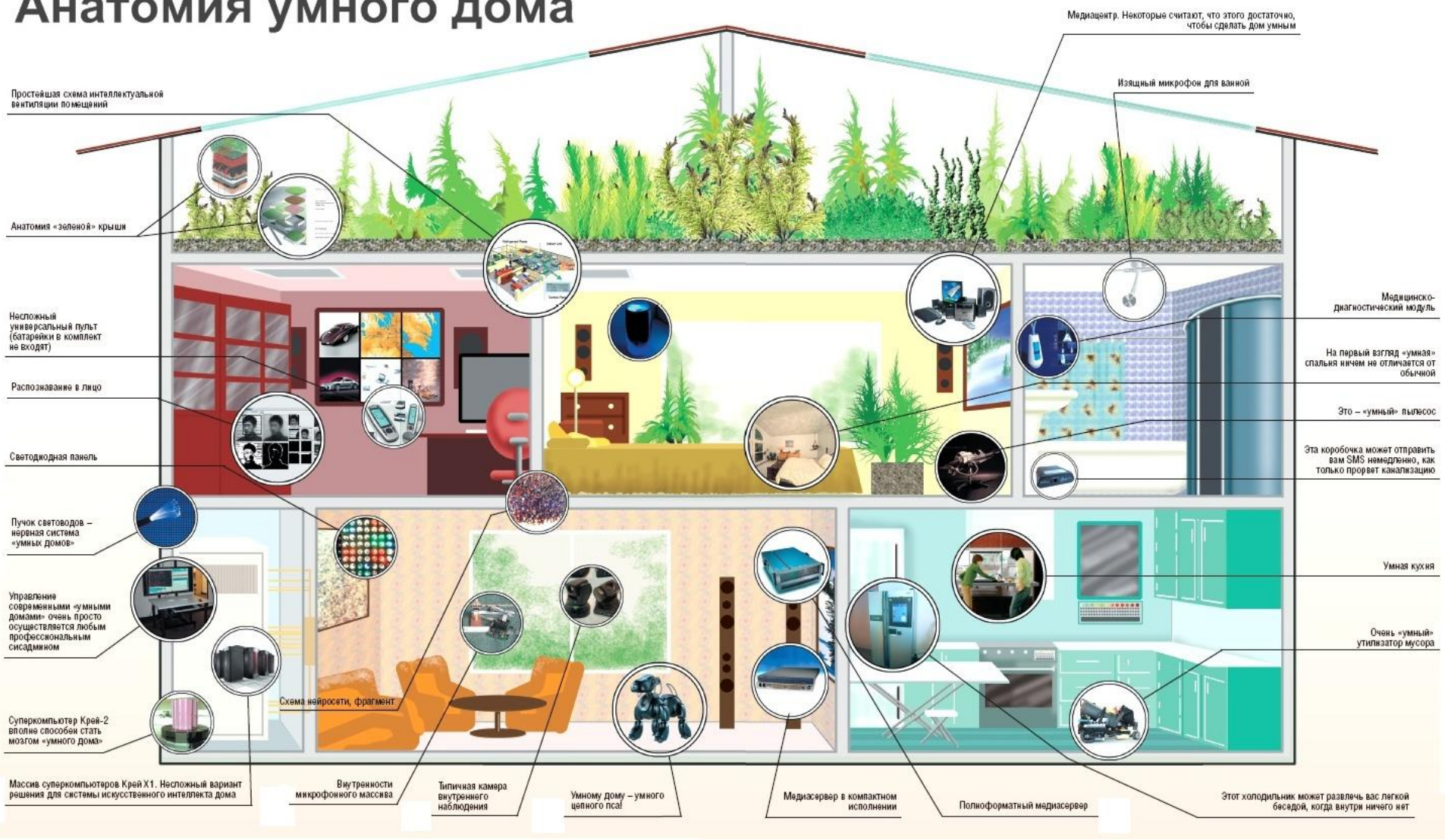
- Ориентация разрабатываемого языка программирования на предметную область;
- Анализ свойств разрабатываемого языка;
- Построение семантической модели языка в соответствии с предъявляемыми требованиями;
- Выбор схемы (модели) трансляции и исполнения, переходу к более формализованным моделями описания синтаксиса и семантики и т. д.

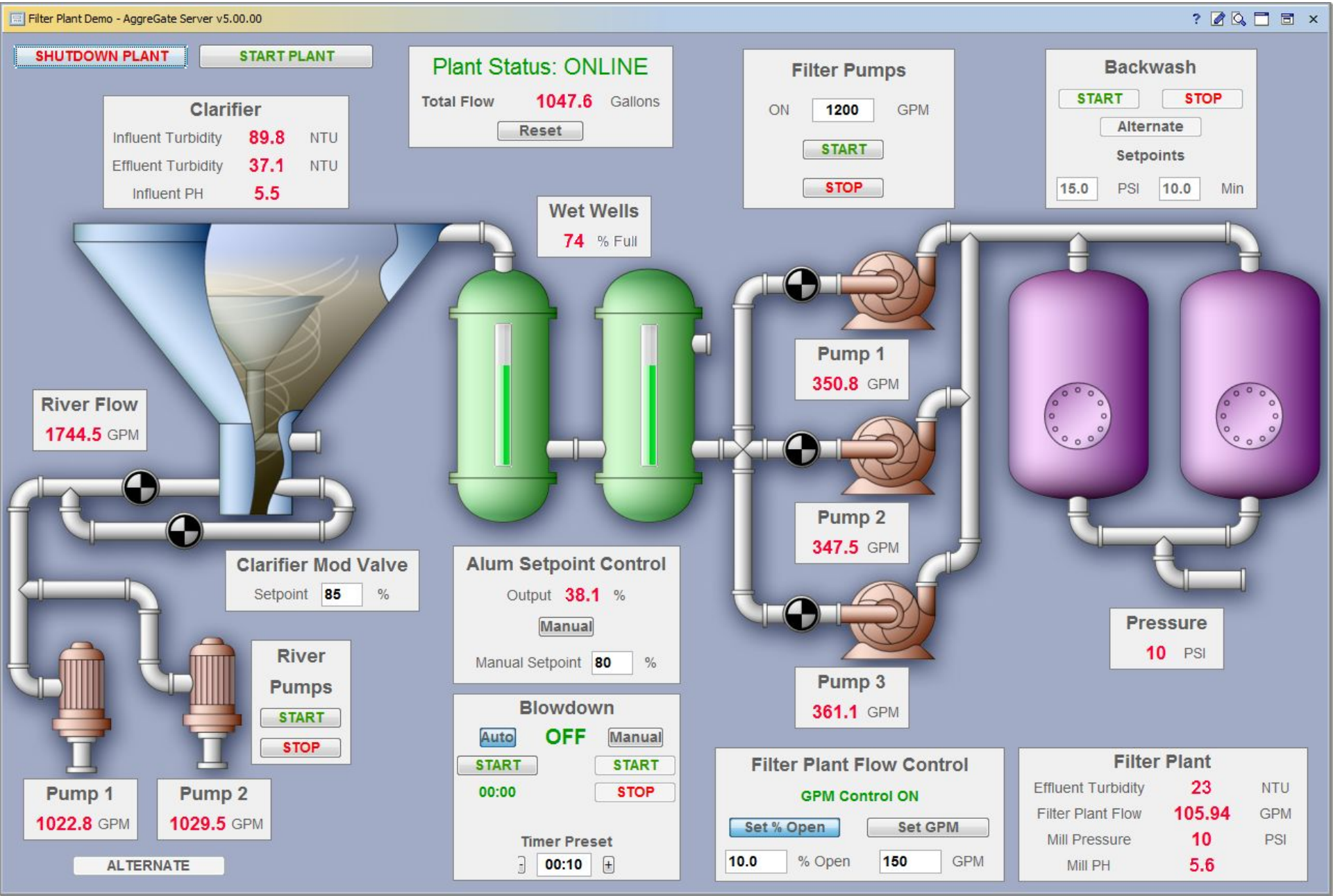
То есть, не только знание теории построения трансляторов

Разработка системы управления технологическим процессом

- Анализ производственного процесса;
- Особенности существующих схем управления производством;
  Формирование схемы (модели) управления;
- Выбор датчиков, системы управления, инструментальных средств;
- и т.д.

Анатомия умного дома



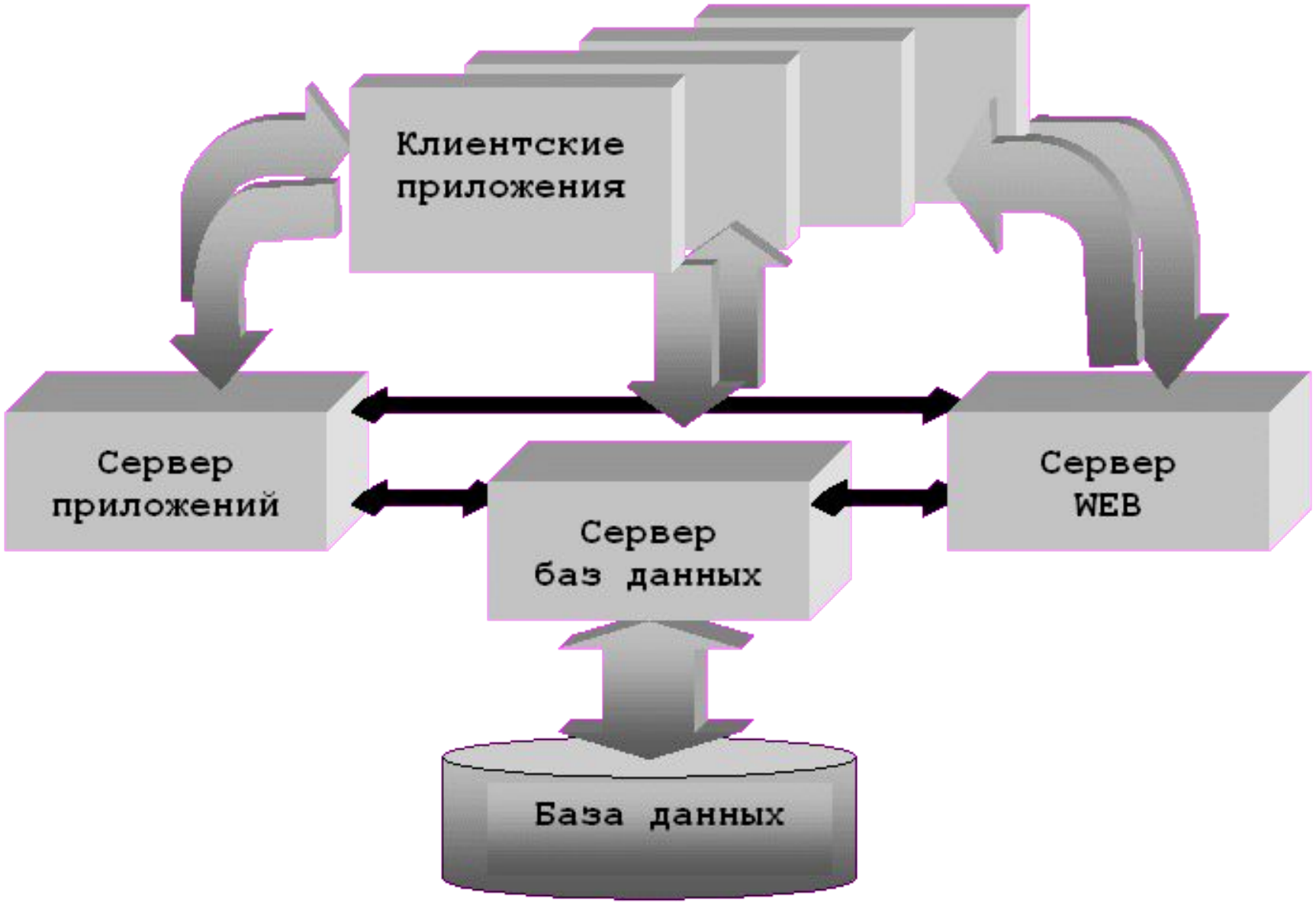

Filter Plant Demo - AggreGate Server v5.00.00
SHUTDOWN PLANT
START PLANT
Plant Status: ONLINE
Total Flow 1047.6 Gallons
Reset
Filter Pumps
ON 1200 GPM
START
STOP
Backwash
START
STOP
Alternate
Setpoints
15.0 PSI 10.0 Min
Clarifier
Influent Turbidity 89.8 NTU
Effluent Turbidity 37.1 NTU
Influent PH 5.5
Wet Wells
74 % Full
River Flow
1744.5 GPM
Pump 1
350.8 GPM
Pump 2
347.5 GPM
Pump 3
361.1 GPM
Pressure
10 PSI
Clarifier Mod Valve
Setpoint 85 %
Alum Setpoint Control
Output 38.1 %
Manual
Manual Setpoint 80 %
River Pumps
START
STOP
Pump 1
1022.8 GPM
Pump 2
1029.5 GPM
ALTERNATE
Blowdown
Auto
OFF
Manual
START
START
00:00
STOP
Timer Preset
00:10
Filter Plant Flow Control
GPM Control ON
Set % Open
Set GPM
10.0 % Open
150 GPM
Filter Plant
Effluent Turbidity 23 NTU
Filter Plant Flow 105.94 GPM
Mill Pressure 10 PSI
Mill PH 5.6

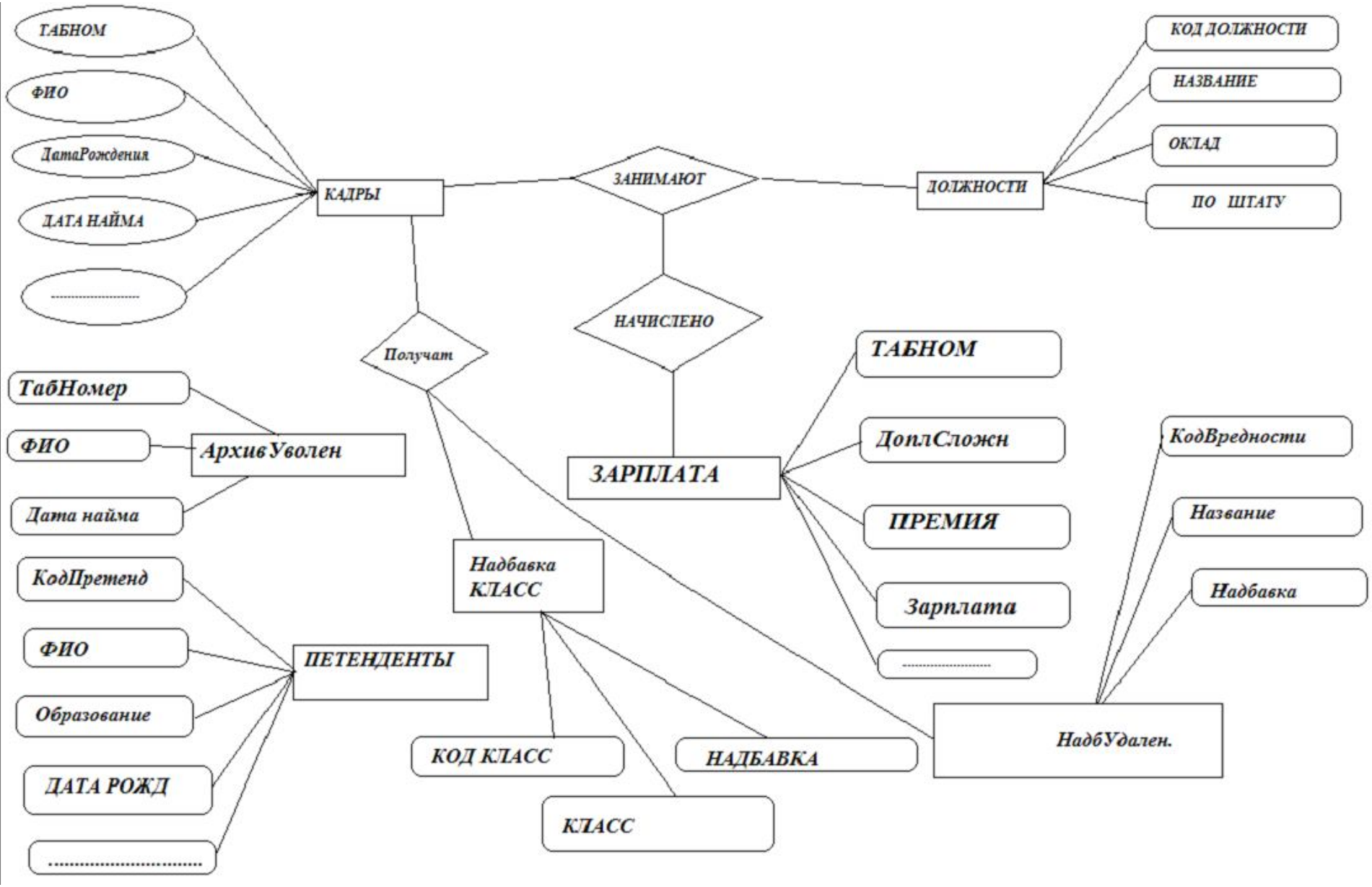

Проект Регистрация Мнемосхемы Тренды/История Сводки События/Аварии Редакторы Прочее... Справка
ВК3.Электродегидраторы ЭД-I/1,2
Контроль аппаратуры
Состояние узлов
0 GLOBAL Норма
1 ARMT1 НЕТ УЗЛА
2 ARMT2 Норма
3 K111 НЕТ АДМИНИСТР
4 K112 НЕТ УЗЛА
5 K121 НЕТ УЗЛА
6 K211 НЕТ УЗЛА
7 K212 НЕТ УЗЛА
8 K213 НЕТ УЗЛА
9 K221 НЕТ УЗЛА
10 --свободно--
11 --свободно--
Тренд ВК4 (ЭД-II/2)
Node 2
ЭД-I/1
ЭД-I/2
Сводка отключений
Обновить
Сводка точек имеющих отключения
N Индекс Имя Отключения Комментарий
1 1286 FC318_INP HARD F пром.воды
Исторические данные с узла ARMT2
Просмотр Обновление Установка времени Календарь Тип выборки
Имя Комментарий
PC2001_INP P речвода от T-12,13 в
LC404_INP L в емкости E-103
F307 F на трубопр.речводы
F355 F воды от T-12,13 в E-10
T1003 T в емкости E-109
Дата Время Данные Флаги
03/09/2001 16:50:07 2.500000 HENE
03/09/2001 16:52:07 0.821642
03/09/2001 16:52:08 2.500000 HENE
Настройка регулятора
FC321K1
Шкала 0...63 м3/ч
Применить
10 минут
Автомат
Коэффициент
Основной
0.70
В зоне
0.00
Время интегр.
30.00
Время дифференц.
0.00
Зона нечувствит.
0.30
Задание уст.
Вход
Выход
Контрол
6.91545
0.0000
Описание объекта
Узел(0) 'GLOBAL' Класс(1) 'ANALOG'
Норма
Наименование Значение
Имя объекта LC406_INP
Комментарий L раздела фаз в ЭД-I/1
Тип данных ВЕЩЕСТВЕННЫЙ
Накопители
Файл Редактирование
2001
Сентябрь
Вс Пн Вт Ср Чт Пт Сб
Стандартные интервалы
За выбранный день
Дополнительные интервалы
Список накопителей
N Индекс Имя Ед.изм Вкл
1 001 FC317K1_INP [0..63]м3/ч вкл
2 897 FC318K1_INP [0..63]м3/ч вкл
3 456 TS136K1_INP [0..100]град.С вкл
4 484 PS188K1_INP [0..0.63]КПа вкл
5 2204 PS2027K3_INP [0..0.4]МПа вкл
Данные за выбранный интервал
Суточные данные за 03 Сентябрь 2001 г.
N Сумма Среднее Мин. Макс. Время
08 нет данных 000 00:00
09 нет данных 000 00:00
10 нет данных 000 00:00
11 нет данных 000 00:00
12 нет данных 000 00:00
13 нет данных 000 00:00
14 нет данных 000 00:00
15 нет данных 000 00:00
16 3.05182 20.3454 0 63 000 00:09
17 нет данных 000 00:00
18 нет данных 000 00:00
19 нет данных 000 00:00
20 нет данных 000 00:00
21 нет данных 000 00:00
22 нет данных 000 00:00
Функциональные клавиши
Файл
Код Комментарий Действие
0xff0b9 9 grey Выключить дискретную точку
0x0032 2 Включить дискретную точку
0x0032 2 Выключить дискретную точку BL1_KO
0x0036 6 Вызов набора трендов Trend_VK3__ED_I_1_.trd
0xff0c1 f4 Вызов набора трендов Trend_VK1__T_14_T_7_1_.trd
0xff050 home Вызов набора трендов Trend_VK2__T_12_T_13_T_8_1_2__T_7_2_.trd
0xff054 down Вызов набора трендов Trend_VK3__ED_I_1_.trd
0x0020 space Выключение звука 1_2/VK13_Pokazateli_toka_i_naprjazh_elektrod.mn
0xff014 scroll Вызов мнемосхемы 1_2/VK22_Blokirovochnye_par_ry_ED_III_2.mn
03/09/2001 17:02:15 -2793 ARMT2 ASU Переход через границу Q507 1.000000 Анализ азота на содерж.кислор HENE

Разработка информационных систем

- Анализ предметной области; Построение модели предметной области;
- Формирование моделей данных, работ, процессов, потоков;
- и так далее.


Департамент образования региона
Центральное информационное хранилище
Отчеты
НСИ, запросы и формы отчетов
Городские органы управления образованием регионального центра
НСИ, запросы и формы отчетов
Отчеты
НСИ, запросы и формы
Органы управления образованием городов и районов региона
Районные органы управления образованием регионального центра
Отчеты
Отчеты
НСИ, запросы и формы отчетов
НСИ, запросы и формы отчетов
Образовательные учреждения различных типов и видов
Образовательные учреждения различных типов и видов
НСИ, запросы и формы отчетов

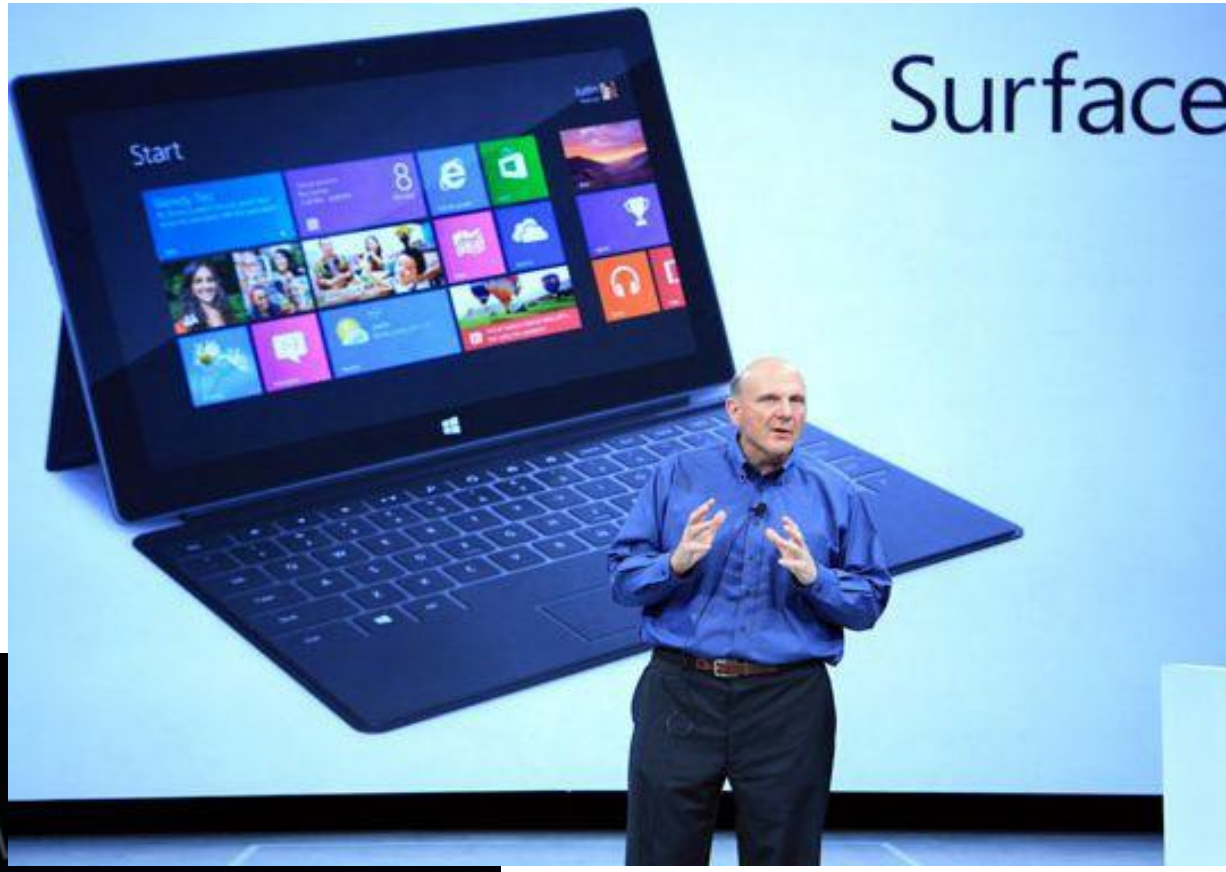

Клиентские приложения
Сервер приложений
Сервер баз данных
Сервер WEB
База данных


ТАБНОМ
ФИО
ДатаРождения
ДАТА НАЙМА
.....
КАДРЫ
ЗАНИМАЮТ
ДОЛЖНОСТИ
КОД ДОЛЖНОСТИ
НАЗВАНИЕ
ОКЛАД
ПО ШТАТУ
НАЧИСЛЕНО
Получат
ТабНомер
ФИО
АрхивУволен
Дата найма
ТАБНОМ
ДоплСложн
ЗАРПЛАТА
ПРЕМИЯ
Зарплата
.....
КодВредности
Название
Надбавка
Надбавка КЛАСС
КодПретенд
ФИО
Образование
ДАТА РОЖД
.....
ПЕТЕНДЕНТЫ
КОД КЛАСС
КЛАСС
НАДБАВКА
НадбУдален.

Влияние особенностей архитектур вычислительных систем («железо», ОС, ЯП)

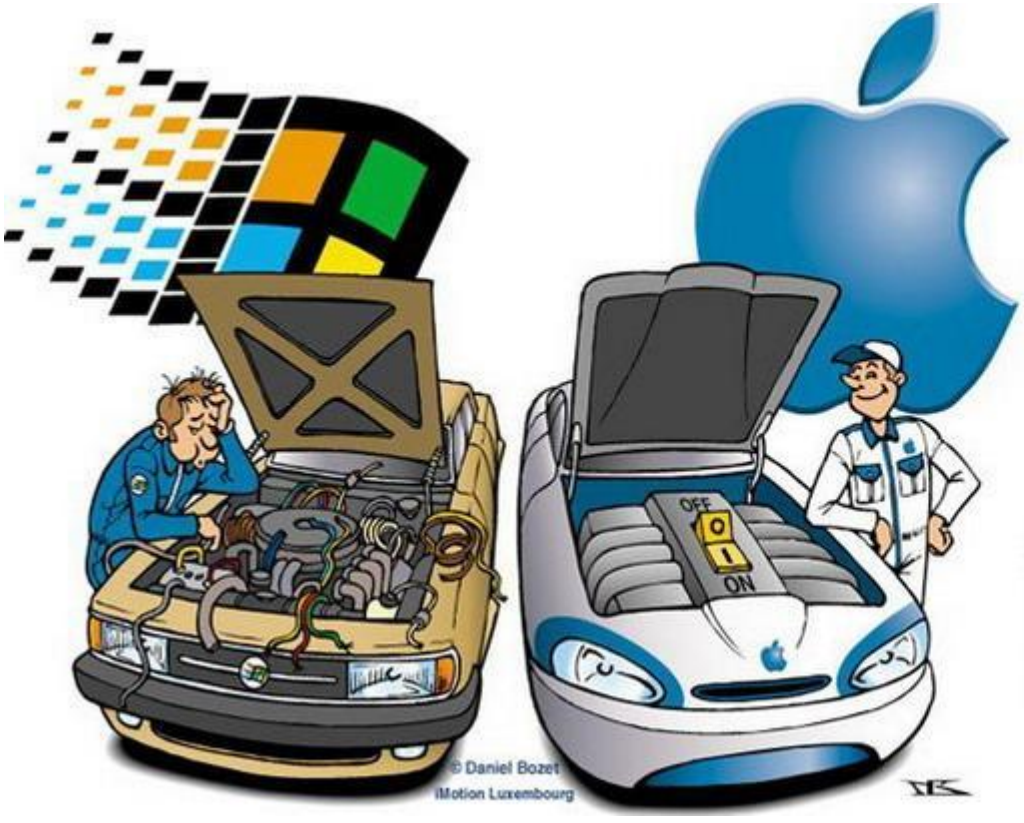
- время выполнения программы
- особенности программирования
- особенности оборудования

...


Start
Surface


The new


Linux vs. Windows


OFF
I
ON
© Daniel Bozet
iMotion Luxembourg

Наряду с проблемами увязки моделей предметной области и архитектуры, встают проблемы, определяемые **спецификой инструментов, используемых для создания программ**.

Выделилось технологическое направление, напрямую не связанное с предметной областью. В его рамках формулируются требования:

- к средствам кодирования, обеспечивающим написание программ,
- к средствам проектирования, определяющим переход от моделей предметной области к программам.

В рамках технологии программирования проводятся исследования методов разработки, обеспечивающих создание продуктов, соответствующих заданным **критериям качества**.

Ряд критериев вытекает из особенностей построения моделей предметной области. Другие обуславливаются сугубо внутренними причинами.

Вместе они характеризуют комплекс проблем, преодолеть который пытаются разработчики программного обеспечения.

Критерии качества программного обеспечения

Корректность (правильность). Обеспечивает правильную обработку на правильных данных.
Устойчивость. "Элегантное" завершение обработки ошибок.
Расширяемость. Может легко адаптироваться к изменяющимся требованиям.
Многократность использования. Может использоваться и в других системах, а не только в той, для которой было создано.
Совместимость. Может легко использоваться с другим программным обеспечением.

Эффективность. Эффективное использование времени, компьютерной памяти, дискового пространства и т.д.

Переносимость (мобильность). Можно легко перенести на другие аппаратные и программные средства.

Верификация. Простота проверки, легкость разработки тестов при обнаружении ошибок, легкость обнаружения мест, где программа потерпела неудачу, и т.д.

Поддержка целостности. Защищает себя от неправильного обращения и неправильного употребления.

Легкость использования. Для пользователя и для будущих программистов

Невозможно сопоставить важность указанных характеристик, так как все они должны учитываться при разработке программного обеспечения.

Расстановка приоритетов осуществляется в зависимости от целей процесса разработки ПО

Цели и задачи процесса разработки

Разработка ПО - многоэтапный процесс, зависящий от многих факторов, например:

- **видов решаемых, задач**, определяющих содержание создаваемых программ;
- **методологий**, задающих особенности организационного и технического проведения основных этапов разработки ПО;
- **методов и парадигм программирования**, обуславливающих стили кодирования и архитектуры виртуальных машин;
- **аппаратных и системных программных средств**, предоставляющие логические и физические ресурсы для использования ПО.

Цель процесса разработки – создание программы, обеспечивающей решение поставленной задачи некоторым исполнителем.

Решаемая задача описывается совокупностью формальных и эмпирических (неформальных) **моделей**, определяющих как протекающие **процессы**, так и используемые при этом **данные**.

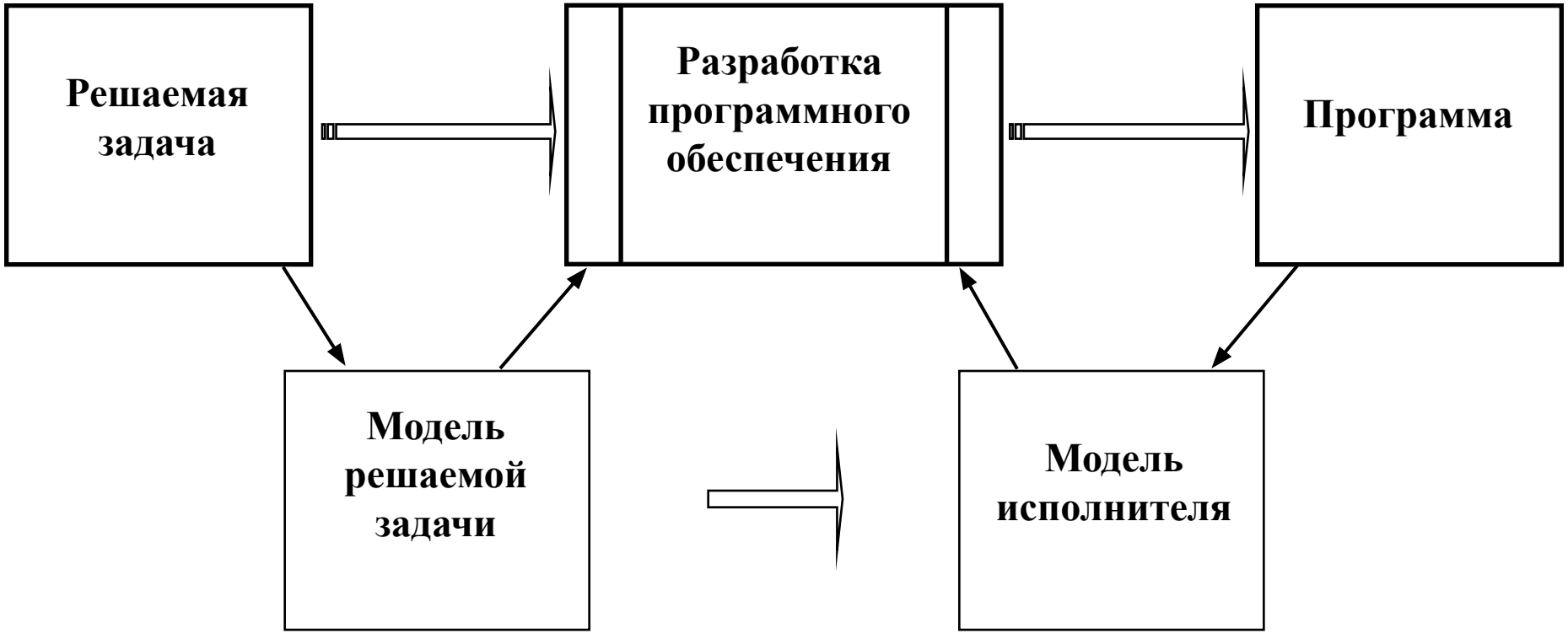
Модель задачи – совокупность специализированных моделей, описывающих различные аспекты решаемой задачи, отражаемые в разрабатываемой программе.

Специализированная модель – модель, предназначенная для описания определенных параметров рассматриваемого явления. Используется для акцентирования внимания на частных характеристиках.

Разрабатываемая программа должна обеспечивать выполнение функций, необходимых для решения задачи, в соответствующем **исполнителе** (вычислительной системе), специфика которого отражается в его **модели**.

Модель исполнителя – совокупность специализированных моделей, описывающих организацию и поведение вычислительной системы, осуществляющей выполнение программы.

Создаваемая программа является отображением модели решаемой задачи на модель исполнителя

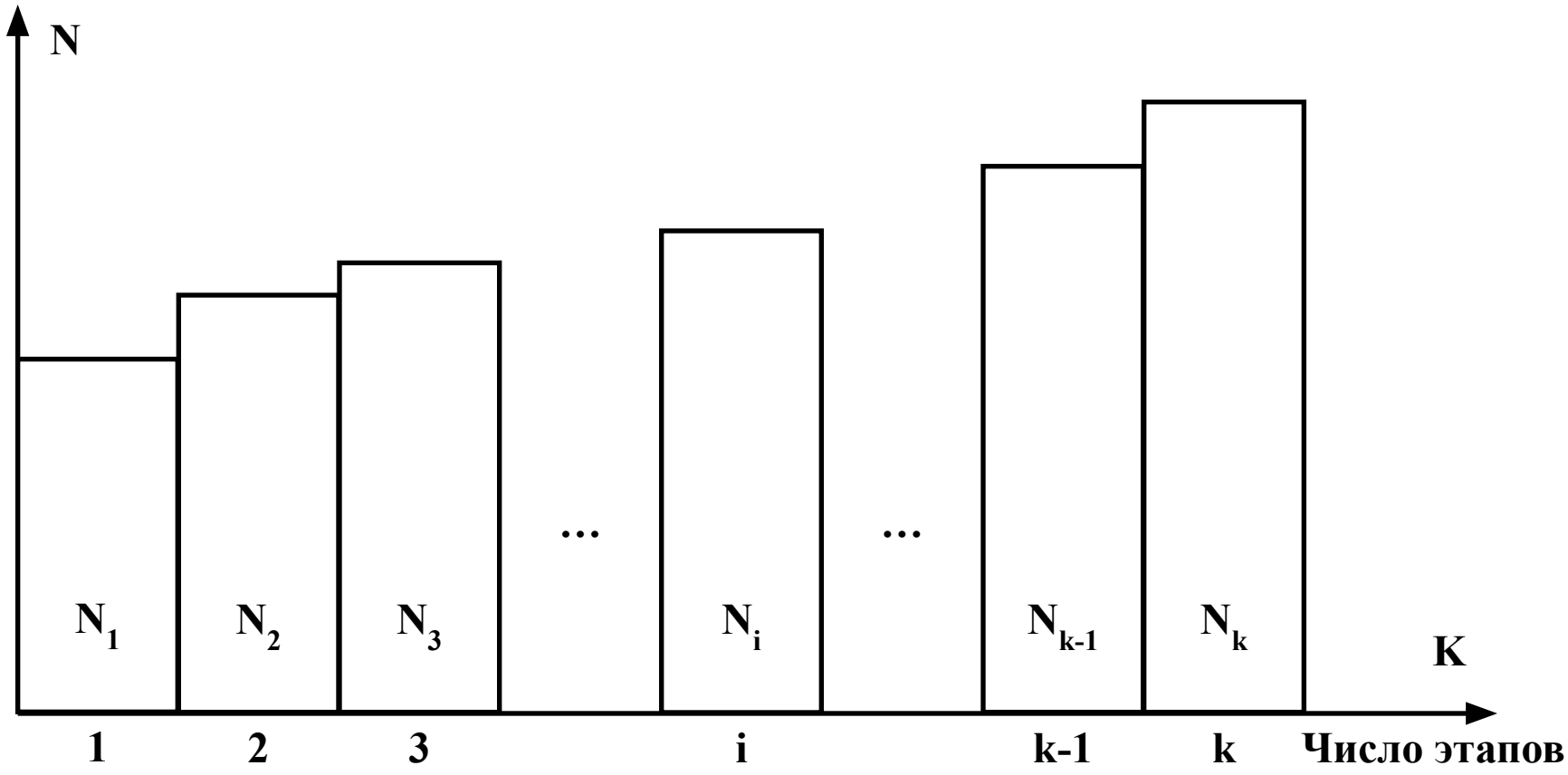


Процесс разработки программного обеспечения как отображение модели задачи на модель исполнителя

Трудоемкость программирования, с одной стороны, определяется **количеством специализированных моделей**, описывающих задачу, их размером, семантическим **отличием от специализированных моделей исполнителя**.
С другой стороны она зависит от **характеристик исполнителя**, задающего требования к уровню абстракции разрабатываемой программы и ее приближенностью к архитектуре реального вычислителя.

Влияние сложности задачи на процесс преобразования исходных моделей на модели исполнителя

Число моделей



Тенденция к нарастанию числа моделей с каждым последующим этапом

Необходимость в преобразованиях обуславливается наличием **семантического разрыва** между моделями задач и исполнителей. Он проявляется в том, что **объекты и операции, которыми разработчик манипулирует при описании задачи, не совпадают с объектами и операциями, используемыми при построении программы**.

Преодоление семантического разрыва осуществляется использованием **методических и технических приемов**, повышающих, к тому же, эффективность процесса разработки ПО.

Методические приемы

Методические приемы ориентированы на формализацию представления моделей и методов перехода между ними.

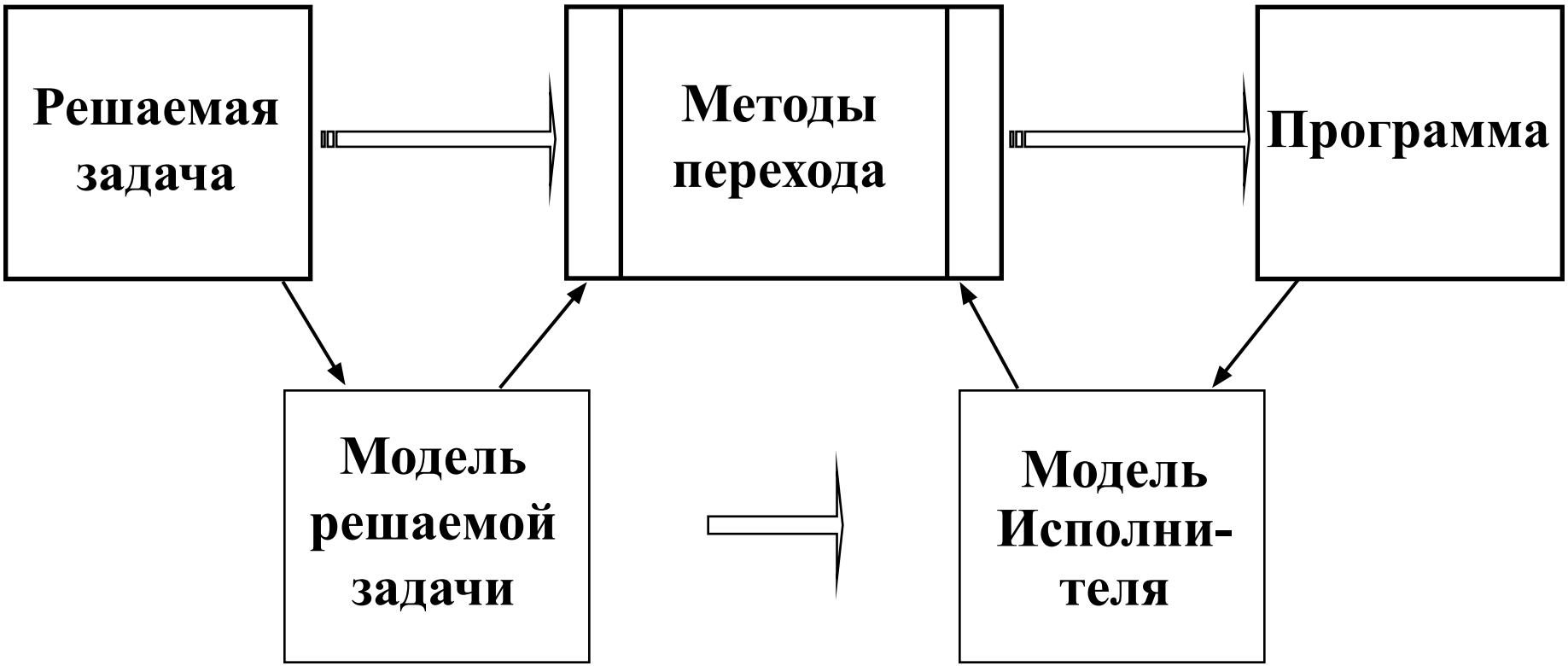
Они позволяют ускорить процесс разработки следующими способами:

- **формализацией предметных областей;**
- **созданием методик разработки программного обеспечения.**

Формализация предметной области

Формализация предметной области заключается в построении ее модели и разработке методов преобразования в модель исполнителя.

Модель предметной области объединяет совокупность специализированных моделей предназначенных для описания определенного класса решаемых задач, что обеспечивают унификацию решения сверху. Дальнейший переход к модели исполнителя обычно осуществляется по выработанным методам или алгоритмам

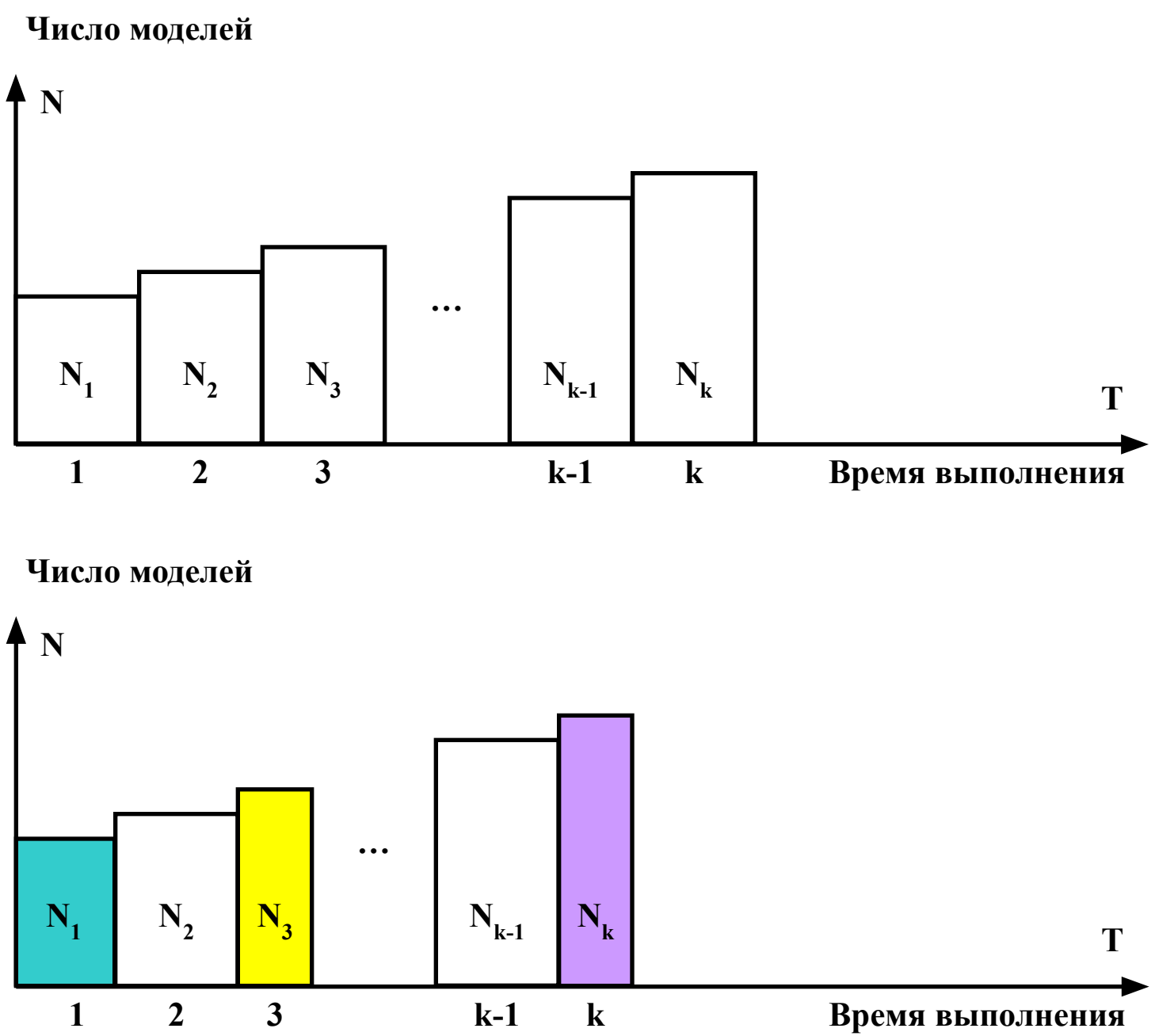


общий механизм перехода с использованием разработанных методов

Подобный подход широко используется при разработке программ в разных предметных областях, например:

1. Построение синтаксических и лексических анализаторов. Модель предметной области описывается с использованием формальных грамматик, эквивалентность между различными грамматиками и автоматами позволяет перейти к распознавателям путем использования наработанных методов их программной реализации.

2. Разработка программных систем на основе теории автоматов. Последующая программная реализация автоматов является хорошо отработанным формальным приемом с применением различных технологий.



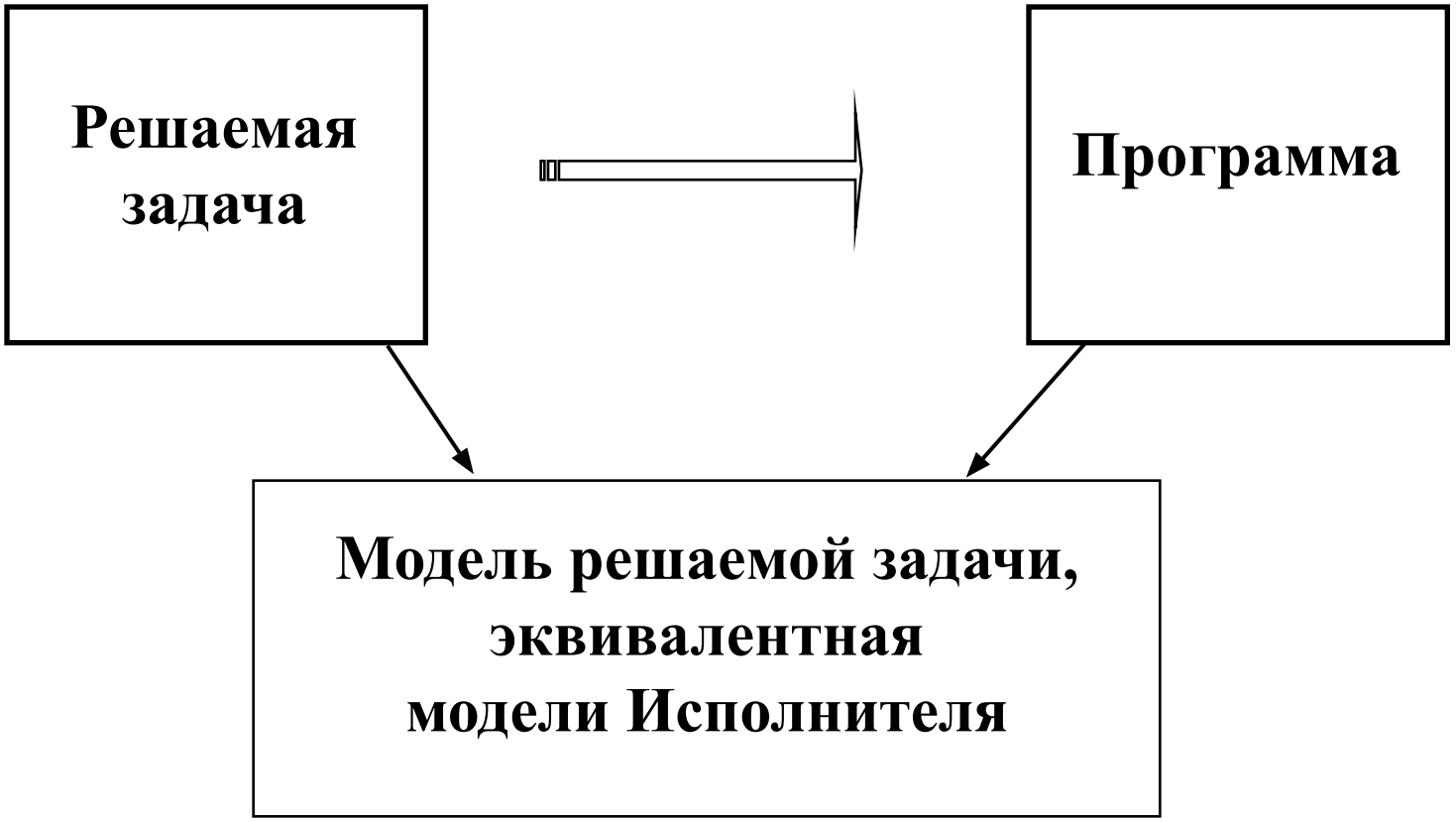
Сокращение длительности проекта при формализации предметных областей

Разработка алгоритмов преобразования одних моделей в другие позволяет автоматизировать процесс и обеспечить представление исходной задачи в виде формализованных данных или программы на **специализированном (проблемно-ориентированном) языке программирования**.

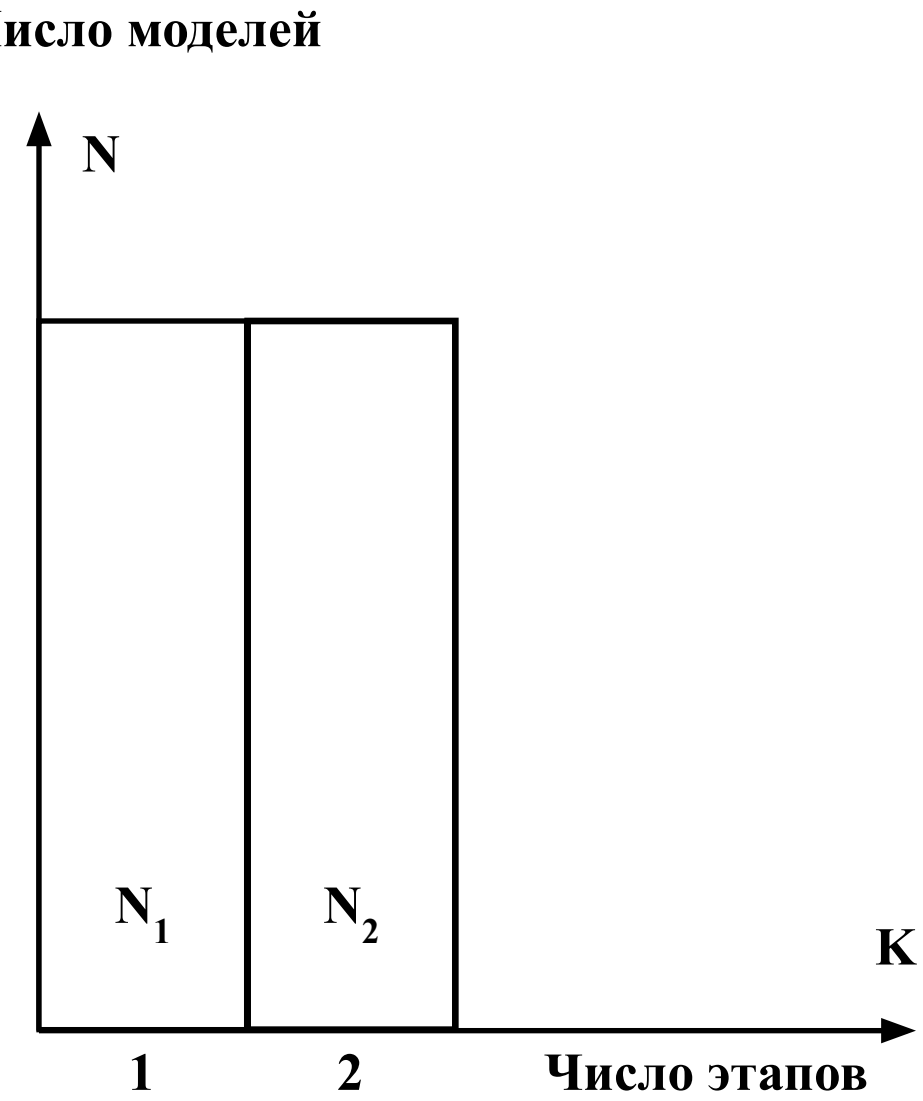
Фактически это означает слияние моделей задачи и исполнителя.

Примеры:

- многие языки имитационного моделирования;
- языки, для управления оборудованием.
- системы генерации лексических и синтаксических анализаторов языков программирования по описанию синтаксиса на соответствующих метаязыках



слияние моделей при замене методов перехода системой программирования



Использование специализированных языков (идеальный вариант)

Достоинства подхода:

1. более высокая скорость разработки (к программированию можно приступать во время анализа задачи);

2. программирование в терминах предметной области, что еще больше повышает эффективность процесса разработки (программирование без программирования).

Недостатки:

1. отсутствие гибкости (предметная ориентация моделей не позволяет непосредственно использовать накопленные методы и инструменты в других областях);
2. ориентация на достаточно узкую категорию задач (относительно простых);
3. необходимость разработки и использования специализированных инструментальных средств.

Резюме:

Эффективно при решении достаточно простых задач узкого класса. Увеличение размерности резко повышает количество применяемых специализированных моделей, **пригодных для использования в разнообразных предметных областях**. Это ведет к увеличению сложности методов формализации и **уменьшению эффективности комплексного использования специализированных моделей**.

В подобных случаях целесообразнее переходить к универсальным методам разработки ПО.

Близкий пример: **вычислить 100!**

1. Алгоритм
2. Язык программирования
3. Программа

Алгоритм 1:

n! = если i=1 то 1
иначе n*(n-1)!

Алгоритм 2:

$$n! = \prod_{i=1}^{n} i$$

(между алгоритмами нет особой разницы)

Язык программирования: C++
Программа:

```
#include <iostream>
using namespace std;

typedef unsigned short us;
typedef unsigned long   ul;

// Получение частного и остатка от деления
// вектора коротких целых на 10
int div_mod_10(us v[], int len, int &mod_rez) {
    ul acc;
    int i_v = len - 1;
    mod_rez = 0;

    while(i_v >= 0) {
        acc = (mod_rez << 16) + v[i_v];
        v[i_v] = acc / 10;
        mod_rez = acc % 10;
        --i_v;
    }
    if(v[len-1] == 0) --len;
    return len;
}
```

```
// Вывод в десятичном виде содержимого вектора
коротких целых чисел
void vector_out(us v[], int len) {
    int xmod;
    int new_len;
    if(len > 0) {
        new_len = div_mod_10(v, len, xmod);
        vector_out(v, new_len);
        cout << xmod;
    }
}

// Умножение вектора коротких чисел на целое
положительное число.
int mult(us v[], int len, ul y) {
    ul acc, per = 0;
    for(int i = 0; i < len; i++) {
        acc = (ul)v[i] * (ul)y +per;
        v[i] = acc;
        per = acc >> 16;
    }
    if(per) v[len++] = per;
    return len;
}
```

```
// Функция вычисления факториала
// заданной величины
int fact(us v[], int max_len, ul y) {
    v[0] = 1;
    int len = 1;
    for(ul i = 2; i <= y; i++) {
        len = mult(v, len, i);
        if(len >= max_len) {
            cout << "Vector overflow!!!
         << endl;
            exit (1);
        }
    }
    return len;
}
```

```
us rezult[200];

int main() {
    int len = fact(rezult, 200, 100);

    cout << endl << "rezult =  ";
    vector_out(rezult, len);
    cout << endl;
    cout << endl;

    // Контрольный тест с использованием
    // арифметики с плавающей точкой
    long double f = 1;
    for(int n=2; n<=100; n++)
        f *= n;
    cout << "f = " << f << endl;
}
```

```
rezult =
9332621544394415268169923885626670049071596826438162146859296389521759999322991560894146397615651828625369792082722375825118521091686400000000000000000000000

f = 9.33262e+157
```

Язык программирования: Lisp
Программа 1:

```
(defun factorial (n)
  (if (= n 0)
      1
      (* n (factorial (- n 1)))))
```

Программа 2:

```
(defun fact-iter (result counter)
 (if (= counter 0)
     result
     (fact-iter (* counter result)
                (- counter 1))))
(defun factorial (n)
  (fact-iter 1 n)
)
```

Результат:

```
> (factorial 100)
93326215443944152681699238856266700490715968264381621468592963895217599993229915608941463976156518286253697920827223758251185210916864000000000000000000000

> (factorial 200)
78865786736479050355236321393218506229513597768717326329474253324435944996340334292030428401198462390417721213891963883025764279024263710506192662495282993111346285727076331723739698894392244562145166424025403329186413122742829485327752424240757390324032125740557956866022603190417032406235170085879617892222789623703897374720000000000000000000000000000000000000000000000000000
```

Язык программирования: Python

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

def factorial(n):
    if n < 2:
        return 1
    f = 1
    while n >= 2:
        f *= n
        n -= 1
    return f

print (factorial(100))
print (2**100)
print (factorial(200))
print (2**200)
```

```
$ python ./fact.py
```

```
933262154439441526816992388562667004907159682643816214685929638952175999932299156089414639761565182862536979208272237582511852109168640000000000000000000
```

```
126765060022822940149670320537 6
```

```
788657867364790503552363213932185062295135977687173263294742533244359449963403342920304284011984623904177212138919638830257642790242637105061926624952829931113462857270763317237396988943922445621451664240254033291864131227428294853277524242407573903240321257405579566602260319041703240623517008587961789222278962370389737472000000000000000000000000000000000000000000
```

```
160693804425899027554196209234116260252220299378279283530137 6
```

The GNU Multiple Precision Arithmetic Library

http://gmplib.org/

GMP is a free library for arbitrary precision arithmetic, operating on signed integers, rational numbers, and floating point numbers. There is no practical limit to the precision except the ones implied by the available memory in the machine GMP runs on. GMP has a rich set of functions, and the functions have a regular interface.

The main target applications for GMP are cryptography applications and research, Internet security applications, algebra systems, computational algebra research, etc.

GMP is carefully designed to be as fast as possible, both for small operands and for huge operands. The speed is achieved by using fullwords as the basic arithmetic type, by using fast algorithms, with highly optimised assembly code for the most common inner loops for a lot of CPUs, and by a general emphasis on speed.

Язык программирования: C

```
#include <gmp.h>
int main(int argc, char* argv[]) {
    mpz_t result;
    unsigned long int x;
    if(argc !=2)
      x = 100;
    else
      x = atoi(argv[1]);
    mpz_init_set_ui(result, 1UL);

    unsigned long int i;
    for(i = 2; i <= x; i++) {
      mpz_mul_ui(result, result, i);
    }
    gmp_printf("%Zd\n", result);
    return 0;
}
```

Язык программирования: C

```
#include <gmp.h>

int main(int argc, char* argv[]) {
    mpz_t result;
    unsigned long int x;
    if(argc !=2)
      x = 100;
    else
      x = atoi(argv[1]);

    mpz_init(result);
    mpz_fac_ui(result, x);

    gmp_printf("%Zd\n", result);
    return 0;
}
```

Язык программирования: C++

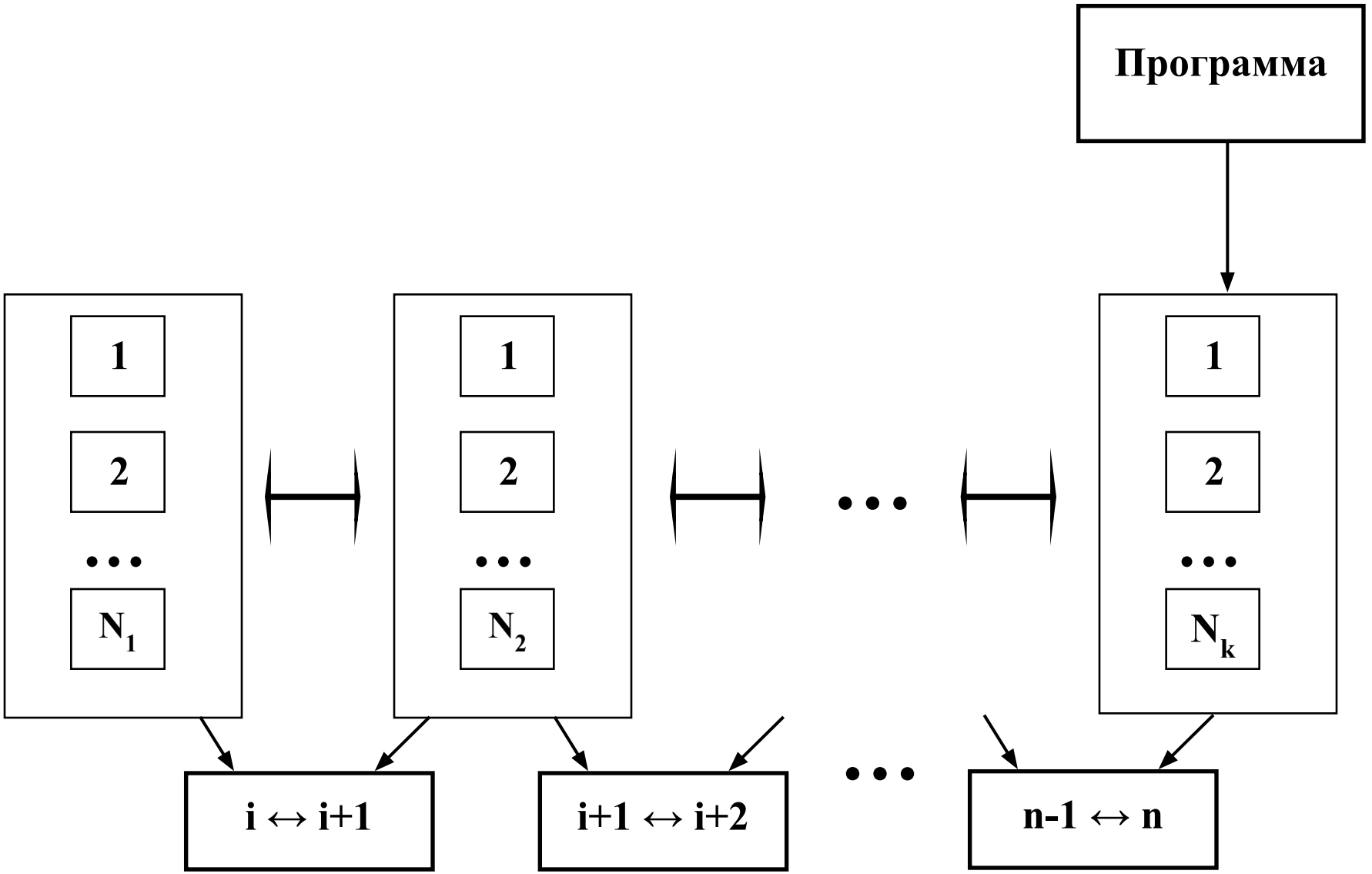
```
#include <stdlib.h>
#include <iostream>
#include <gmpxx.h>
using namespace std;

int main(int argc, char* argv[]) {
    if(argc !=2) {
      cout << "Argumen is absent!!!" << endl;
      return -1;
    }
    mpz_class result(1UL);
    unsigned long int x = atoi(argv[1]);
    for(unsigned long int i = 2; i <= x; i++) {
      result *= i;
    }
    cout << result << endl;
    return 0;
}
```

Методики разработки ПО

Ориентированы на формализацию взаимосвязей между моделями конкретных исполнителей и моделями, используемыми на предшествующих этапах разработки (например, при анализе и проектировании).

Изначальная ориентация Исполнителя на универсальность вычислений обычно предполагает применение методик для широкого класса задач, не связывая их непосредственно с предметными областями.

Они поддерживают процесс преобразования как от модели задачи к модели исполнителя, так и в обратном направлении.

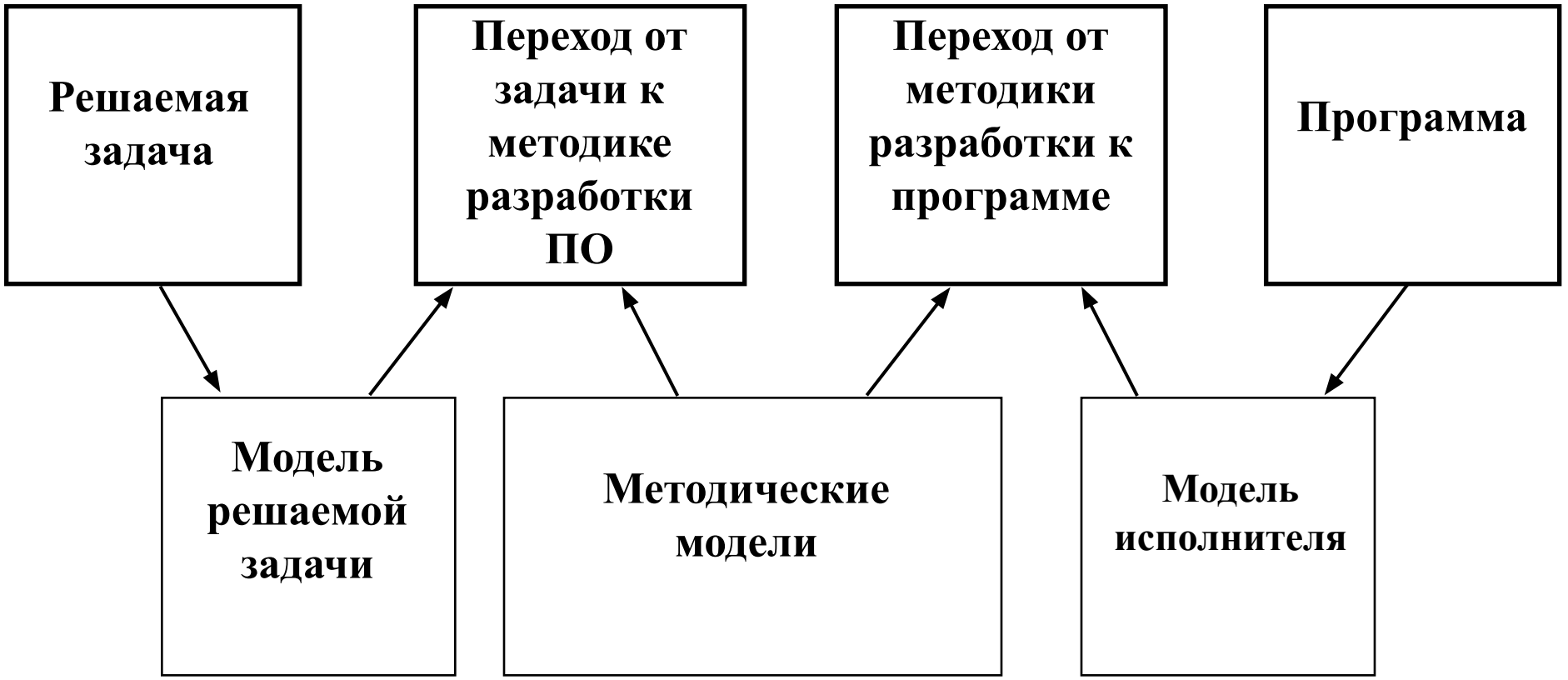

Программа
1
2
...
N1
1
2
...
N2
1
2
...
Nk
i ↔ i+1
i+1 ↔ i+2
n-1 ↔ n

Несмотря на обеспечение прямого и обратного проектирования, основное достоинство методик проявляется в поддержке нисходящей разработки.

Это во многом обуславливается большей наглядностью переходов от универсальных высокоуровневых моделей, ориентированных на описание предметных областей, к моделям, описывающим соответствующих исполнителей.

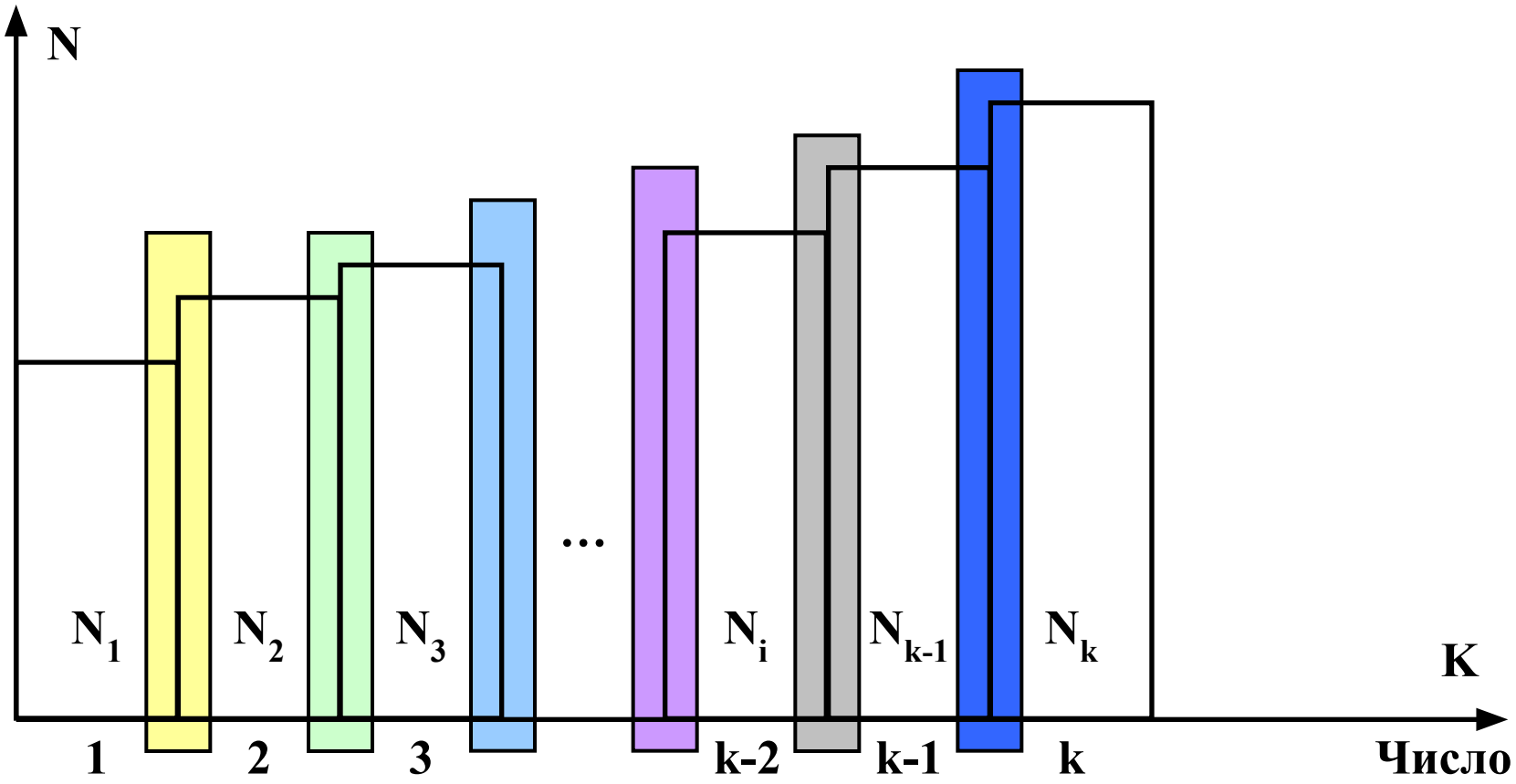
В подобной ситуации **знание задачи повышает эффективность разработки**.

Поэтому, создание программного обеспечения обычно начинается с привязки модели предметной области к моделям анализа и проектирования, предлагаемым используемой методикой.



Совместное использование методов формализации предметной области и методик разработки.

Число моделей



Применение методик (методических моделей)

Достоинства методик разработки ПО:

- универсальность, обуславливающая ориентацию на разработку задач широкого класса;
- поддержку нисходящего и восходящего проектирования;
- поддержку прямого и обратного проектирования;
- возможность использовать инструментальные средства.

Недостатки отдельных методик:

- привязка процесса разработки к определенным методам и исполнителям.

В настоящее время существуют различные методики.
Они являются **составной и неотъемлемой частью методологий разработки программного обеспечения**.

Методологии, наряду с процессами создания программ, дополнительно регламентируют организационную деятельность, анализ, тестирование и сопровождение, что в целом определяет организацию жизненного цикла программы на основе единого концептуального подхода

Примеры методик:
- объектно-ориентированный подход, используемый в составе объектно-ориентированной методологии;
- методы структурного анализа и проектирования, применяемые при разработке информационных систем;
- методы быстрой разработки приложений, ориентированные на построение программ от моделей, определяющих взаимодействие системы с пользователем;
- методика Джексона, используемая для проектирования программы по структурам обрабатываемых данных.

Резюме:

Методики широко используются при разработке больших программных систем, так как повышают эффективность процесса проектирования за счет предоставления достаточно простых и ясных подходов, выработанных на основе эмпирического опыта и теоретических исследований.

Их использование, в сочетании с инструментальной поддержкой, обеспечивает сокращение семантического разрыва между моделями различных задач и исполнителем.

Технические приемы

Технические приемы обеспечивают создание инструментальных средств, поддерживающих различные аспекты разработки ПО.

Можно выделить:
- средства поддержки методических приемов;
- вспомогательные средства;
- системы программирования.

Поддержка методических приемов

Инструментальная поддержка методических приемов обеспечивает компьютерное представление и анализ разрабатываемых моделей, преобразование построенных моделей в другие модели, автоматизацию процесса построения требуемой документации, а также ведения организационной деятельности в ходе разработки ПО.

Повышая эффективность процесса разработки, средства поддержки методических приемов, в то же время, **не определяют сам процесс построения программы**. Их использование предполагает дальнейшую доводку программ с применением инструментов, имеющих более тесную связь с архитектурами вычислительных систем.

Примеры

1. **Системы поддержки спецификаций.** Ранние средства. Во многих из них использовались графические языки, которые в основном играли роль вспомогательного документа. Для написания программы необходимо было вручную осуществлять перевод этих диаграмм в код.

Примеры

2. **CASE-средства (Computer Aided Software Engineering).** Инструменты сгладившие, семантический разрыв между рядом моделей предметной области и кодом, обеспечив непосредственное преобразование, как в прямом, так и в обратном направлении.

CASE-средства поддержки методов структурного анализа и проектирования (SADT – Strucure Analis and Design Technique):
BPwin, **ERwin**, ориентированные на использование диаграмм IDEF (Icam DEFinition) и DFD (Data Flow Diagrams)
Silverrun – полный технологический цикл.
Rational Rose – UML (Unified Modelling Language) – Объектно-ориентированная методология.
И много других…

Вендров А.М. Проектирование программного обеспечения экономических и информационных систем: Учебник. – М.: Финансы и статистика, 2002. – 352 с.

Вспомогательные средства

Вспомогательные средства предназначены для повышения эффективности процессов, не связанных с непосредственной разработкой структуры программы, но, в то же время, сильно влияющих на качество и время разработки.

Примеры:

средства отладки;

средства профилирования и другие.

Специфика вспомогательных средств заключается в работе с уже **готовыми программами**.

Это позволяет в дальнейшем игнорировать их влияние на сам процесс разработки программ.

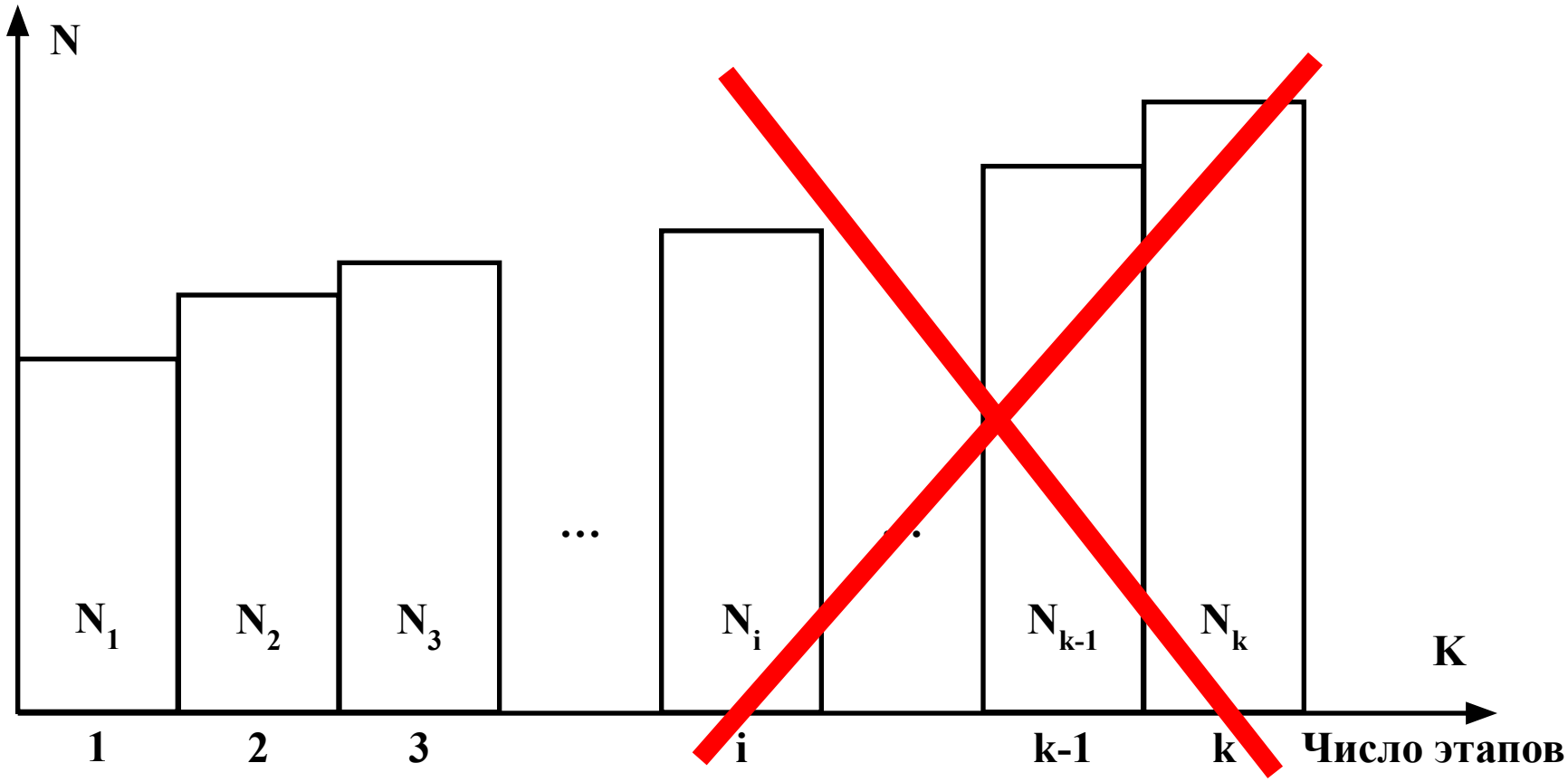
Системы программирования

Системы программирования – инструментальные средства, поддерживающие разработку программ для заданного виртуального исполнителя и их последующее автоматическое преобразование в программы реального исполнителя.

Использование систем программирования позволяет решать задачу повышения эффективности процесса разработки ПО за счет сокрытия исполнителей более низкого уровня, являющихся реальными вычислительными системами.

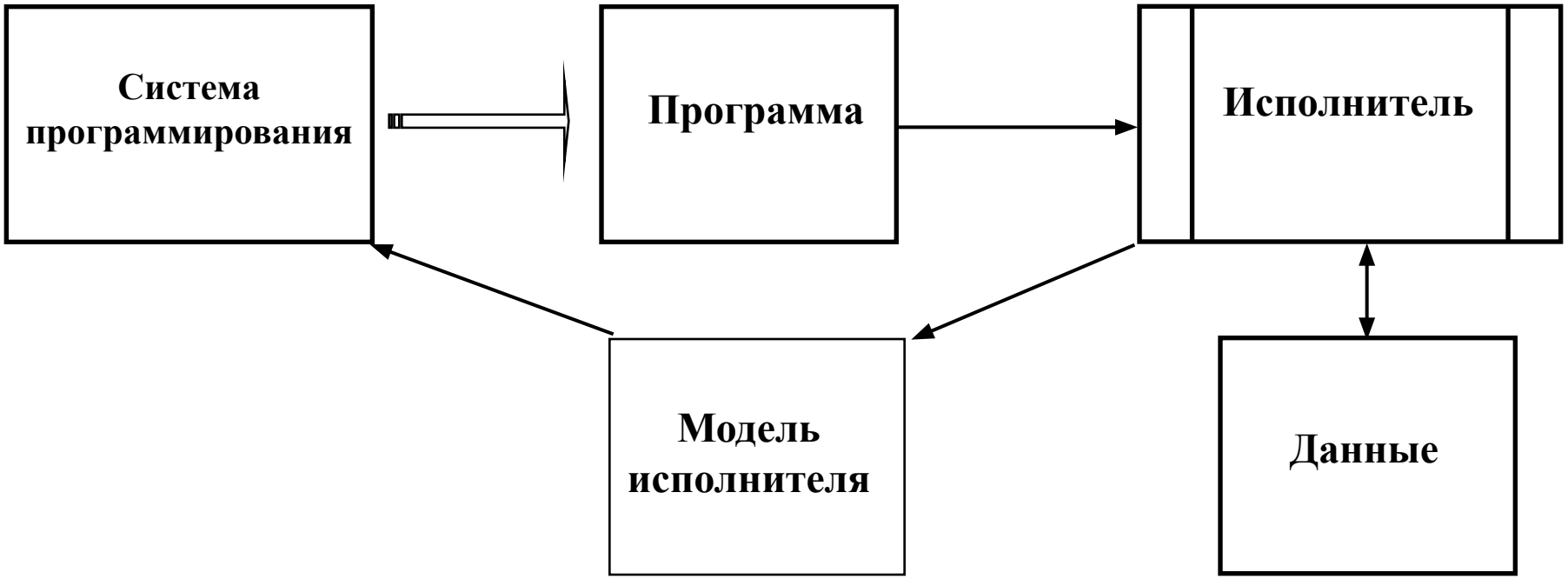
Подход широко используется на практике и позволяет писать программы для **высокоуровневого исполнителя** (определяемого часто как **архитектура виртуальной машины**).

Число моделей



Системы программирования «убирают» часть моделей из процесса разработки

Реальное выполнение полученных программ может осуществляться:
1. Использованием непосредственной интерпретации, полностью скрывающей процесс реального выполнения, который может оказаться намного сложнее.



Непосредственное использование исполнителем программы, написанной с применением системы программирования

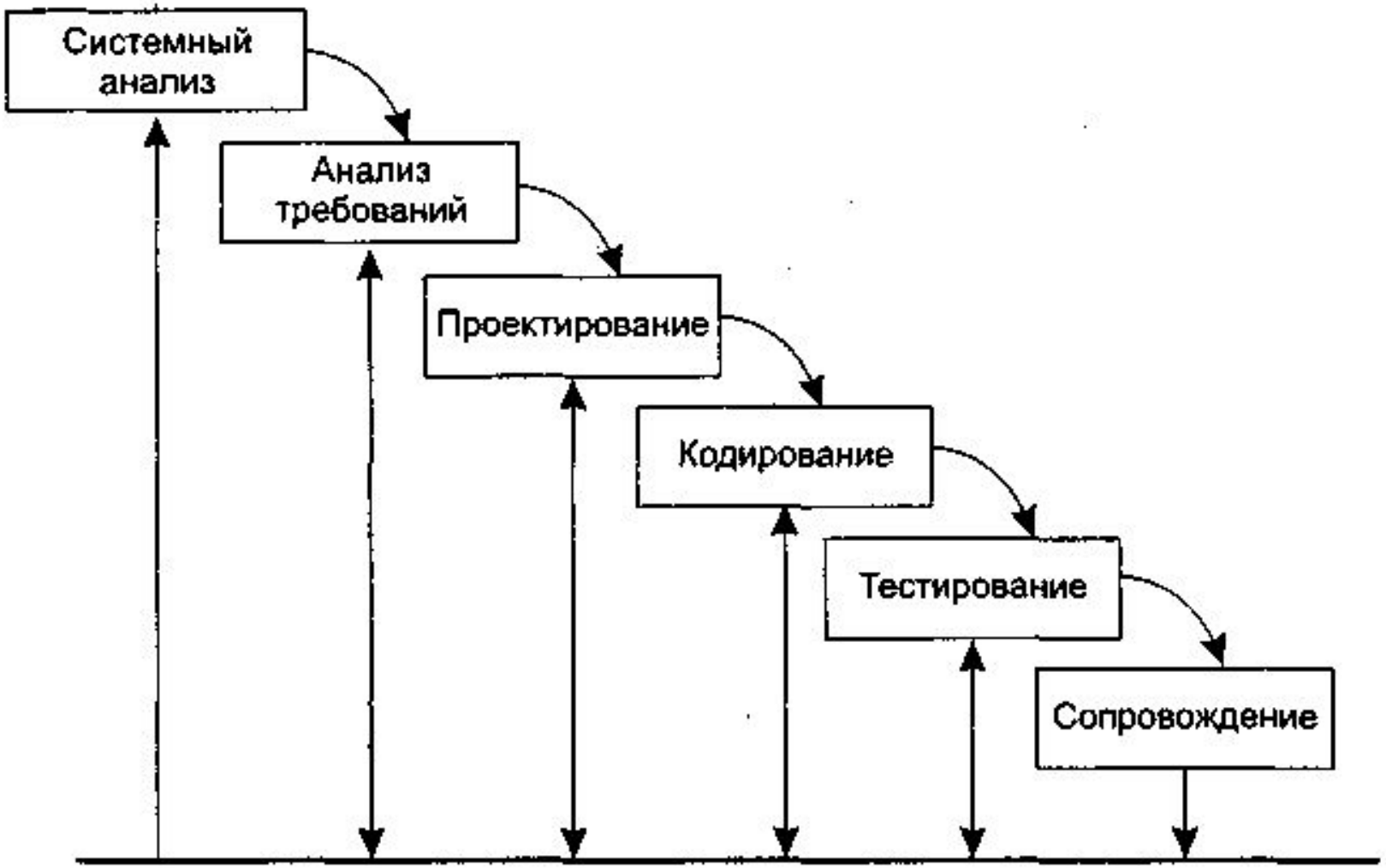
2. Автоматическим и поэтапным преобразованием программы, написанной для виртуальной машины, в программу конечного исполнителя, что позволяет разрабатывать программы для систем, не допускающих непосредственное выполнение (системы компиляции). Преобразование программы может быть многоступенчатым, что может оказаться целесообразным при использовании нескольких промежуточных моделей исполнителей.

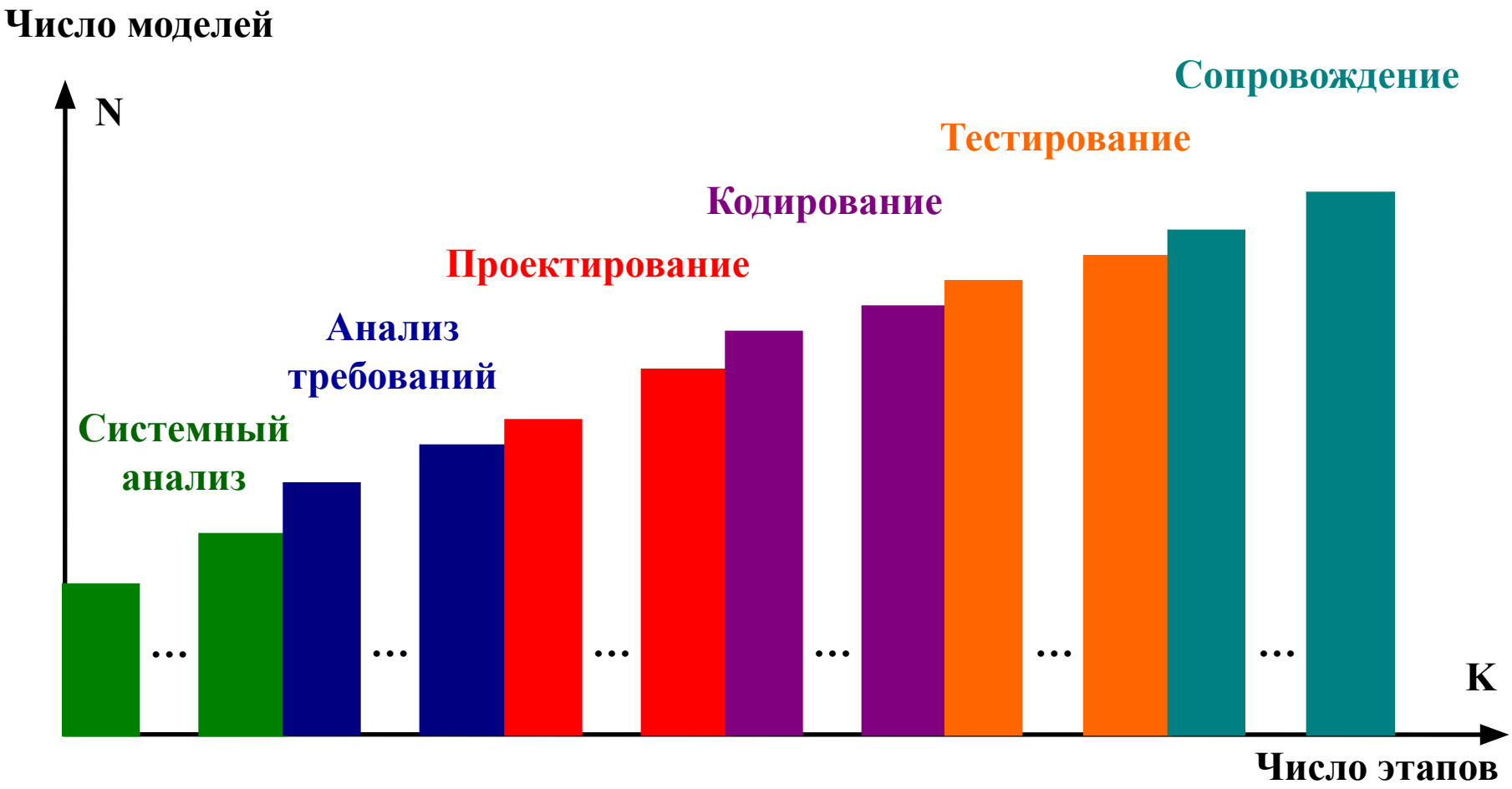
Многоэтапное преобразование модели исполнителя, поддерживаемого системой программирования, на модель исполнителя, реально осуществляющего вычисления

Проводимые промежуточные преобразования могут быть **скрыты от программиста**. Однако знание дополнительной информации о конечном исполнителе позволяет повысить эффективность процессов отладки и выполнения программы.

Исполнители могут отличаться и по степени приближения их моделей к моделям решаемых задач или более высокоуровневым методологическим моделям. Использование в их архитектуре моделей решаемых задач ведет к разработке **специализированных и проблемно-ориентированных систем программирования**. Ориентация на **общие модели проектирования** обуславливает создание **высокоуровневых универсальных средств**. При ориентации на архитектуры **реальных вычислительных систем** исполнитель позволяет разрабатывать **более производительные программы**. Однако процесс разработки ПО усложняется за счет дополнительной детализации.

Представленная модельная специфика конкретизируется при отображении на конкретные процессы разработки ПО…





Связь между моделями и этапами разработки ПО