# Физтех и профессиональное программрование

И. Р. Дединский
Кафедра ТиПИ МФТИ (Parallels, Virtuozzo, Acronis)
Кафедра микропроцессорных технологий ФРТК (Intel)
Кафедра информатики МФТИ

# BINGO!

## Вы поступили на Физтех! Поздравляем!

## Как идет программирование на первом курсе на большинстве факультетов:

- Студентов делят на внутренние группы кафедры информатики (НЕ так, как на других предметах) по результатам контрольной работы на первом занятии
- Занятия во всех группах факультета идут одновременно
- Это позволяет адаптироваться к очень разному уровню первокурсников по информатике
- Между группами можно переходить (в самом начале сентября, только через заявление на кафедру информатики и только через зам. зав. кафедрой по мл. курсам Владимира Константиновича Хохлова, 911 КПМ).

# BINGO!

## Вы поступили на Физтех! Поздравляем!

- **Станете ли Вы программистом? (если хотели)**
  - Не факт =\
- **Почему? Здесь же учат!**
- **Да, учат, и хорошо. Но:**
  - Учат решать отдельные небольшие задачи
  - Существенная часть обучения построена на контестах, а эта модель не очень связана с профессией
  - Нельзя просто взять и «впитать» знания. У профессионального обучения не диффузионная кинетика :)
  - Можно помочь научиться, но заставить стать профессионалом – нельзя.

# Вы поступили на Физтех!

## Посмотрим на проблему шире

- Программистов у нас в стране много, но хороших – мало

- **Почему так?**
  - Школа: все по-разному
    - Информатика – не устоявшийся предмет
    - Ориентация на пользование, а не программирование
  - Дополнительная активность: в основном – олимпиады. Почему?
    - Школам / организаторам выгодно
    - Очень просто использовать стремление соревноваться
    - Достаточно просто организовать, обеспечить массовость

- **В чем проблема?**
  - Профессиональное программирование != соревнованиям
  - Соревнования – это неплохо, но недостаточно для профессии

# Вы поступили на Физтех…

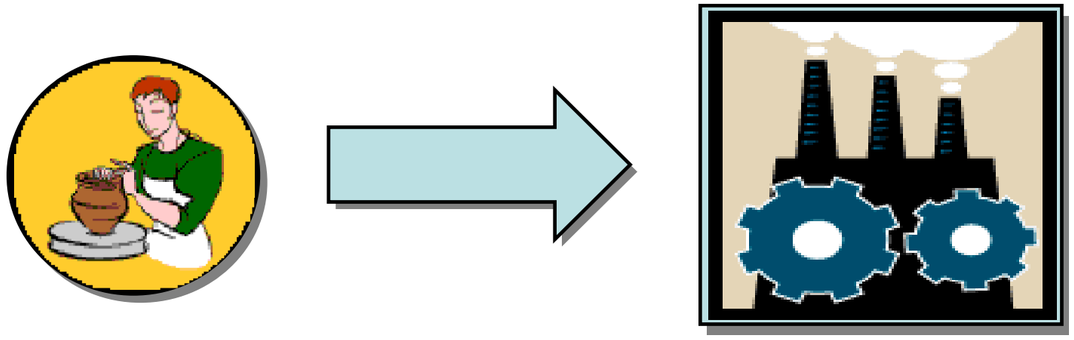
## В чем проблема?

- «Правило 10 000 часов»
  Это около 8-10 лет (в среднем 3-4 часа каждый день)
- Касается любой профессиональной деятельности

## Как быть?

- Надо начинать раньше
- Надо работать правильно
  «Правильно» – это профессионально, независимо от объемов задачи.
  Тогда этот период можно сократить.
- Надо знать, в чем подход профессионала отличается
  от подхода любителя
- Надо это делать не во вред учебе! А то получится
  программист «из Макдональдса». Это непросто на Физтехе.

# В чем особенности профессиональной работы?



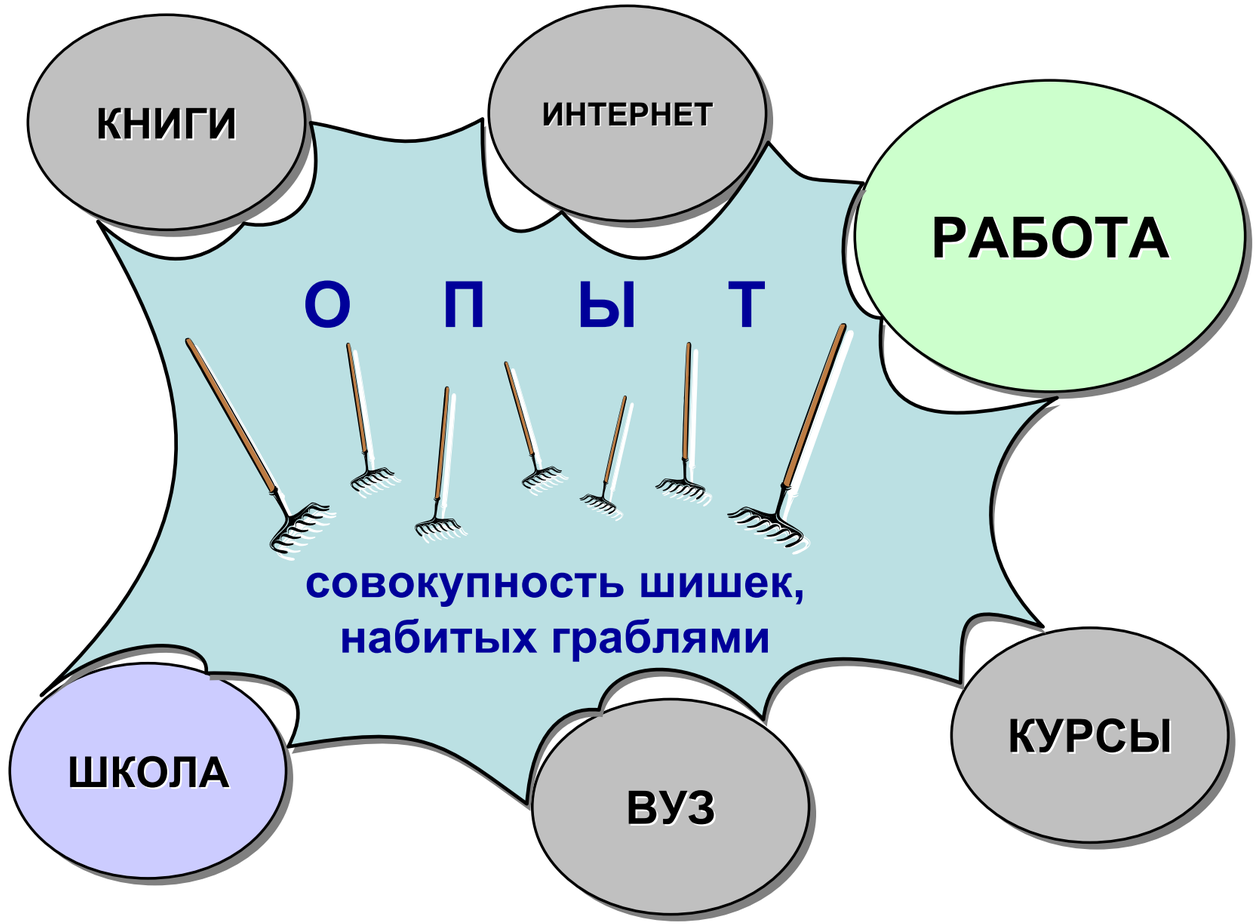
- **Что было раньше? – кустарная модель**
  - Научный стиль разработки (ориентация на метод, алгоритм)
  - Сотрудничество с заказчиком
  - Пользование компьютером предполагало навыки программирования
  - Меньший объем проектов
  - Локальная разработка
  - Преимущественно индивидуальная разработка
  - Небольшое количество стандартов
  - Небольшой объем литературы
  - Культура программирования только развивалась
  - **Легко было учиться самому**

- **Что сейчас? – промышленная модель**
  - Промышленный стиль разработки (ориентация на продукт)
  - Согласование с заказчиком
  - Документирование
  - Ограничения по срокам
  - Большой объем
  - Коллективность
  - Много стандартов, инструментов
  - Чужой и устаревший код
  - Удаленная разработка
  - Большой объем литературы

  - Без культуры программирования работать невозможно!
  - **Учиться самому все труднее и труднее**

# Образовательные разрывы

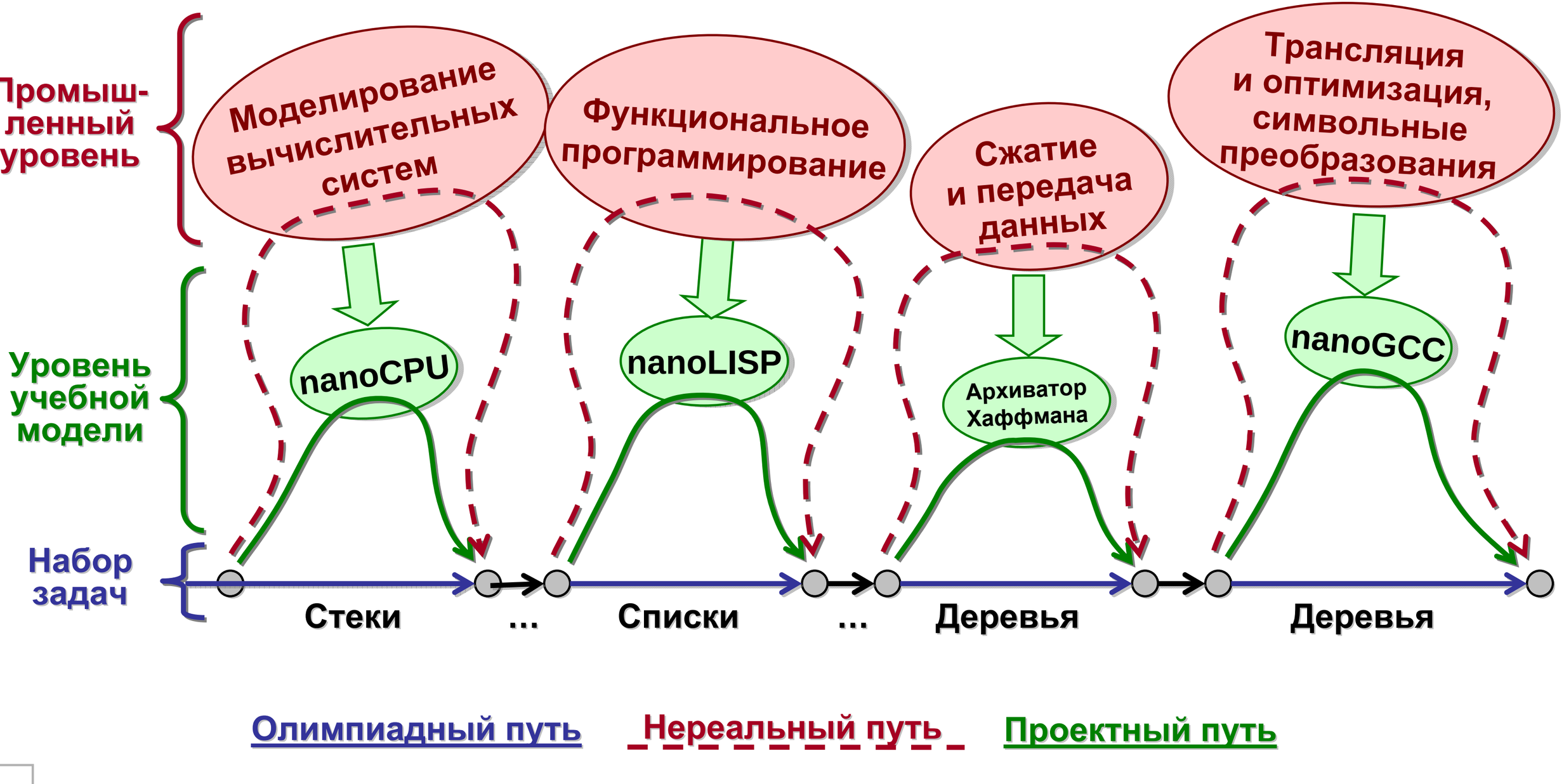
- **Способы преодоления**
  - Самостоятельный метод
  - Олимпиадный метод?
  - Непрерывное профильное образование
- **Повышение КПД образования**
  - Использование ВУЗа как ресурса
- **Для этого нужны:**
  - Активная учебная позиция
    - Мотивация
    - Конструктивное мышление
  - Предшествующий опыт
- **Вывод:**
  - Мало хотеть учиться. Надо уметь хотеть учиться.



# Откуда берутся учебные проекты?

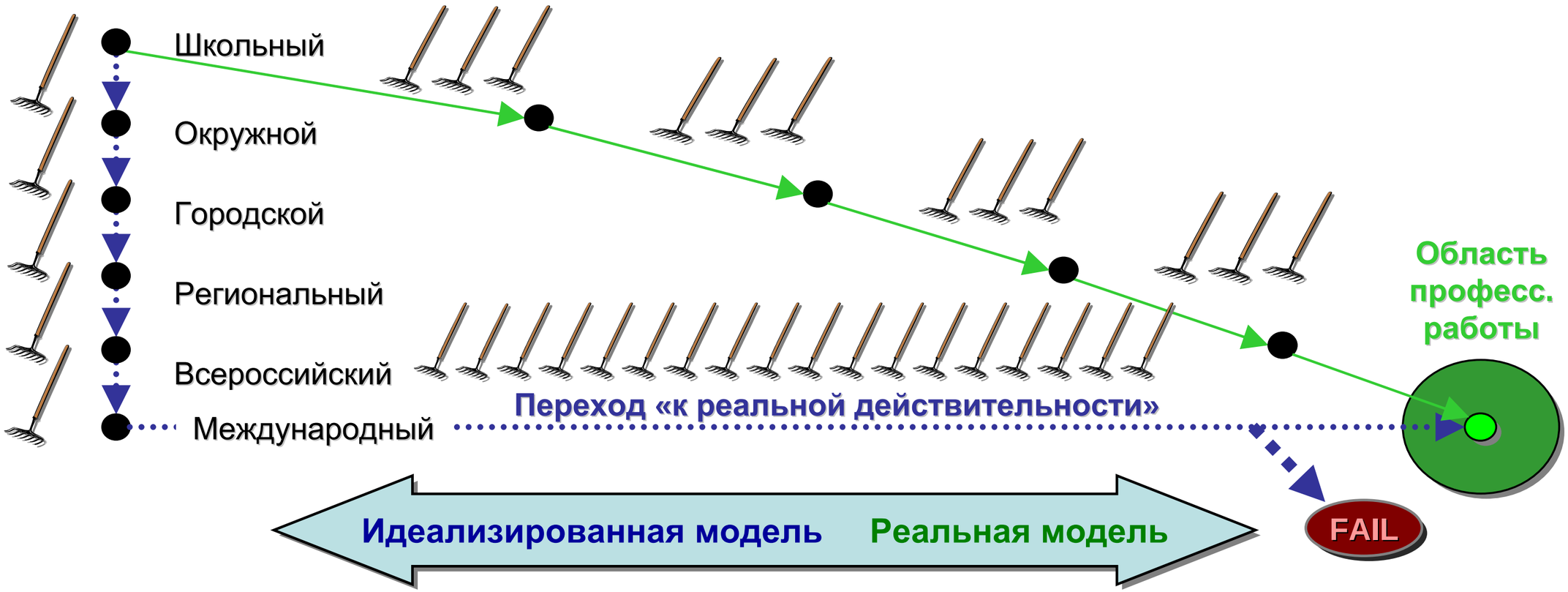
## Пример: курс «Структуры данных и алгоритмы»

(Опущены: очереди, хеш-таблицы, графы, автоматы)



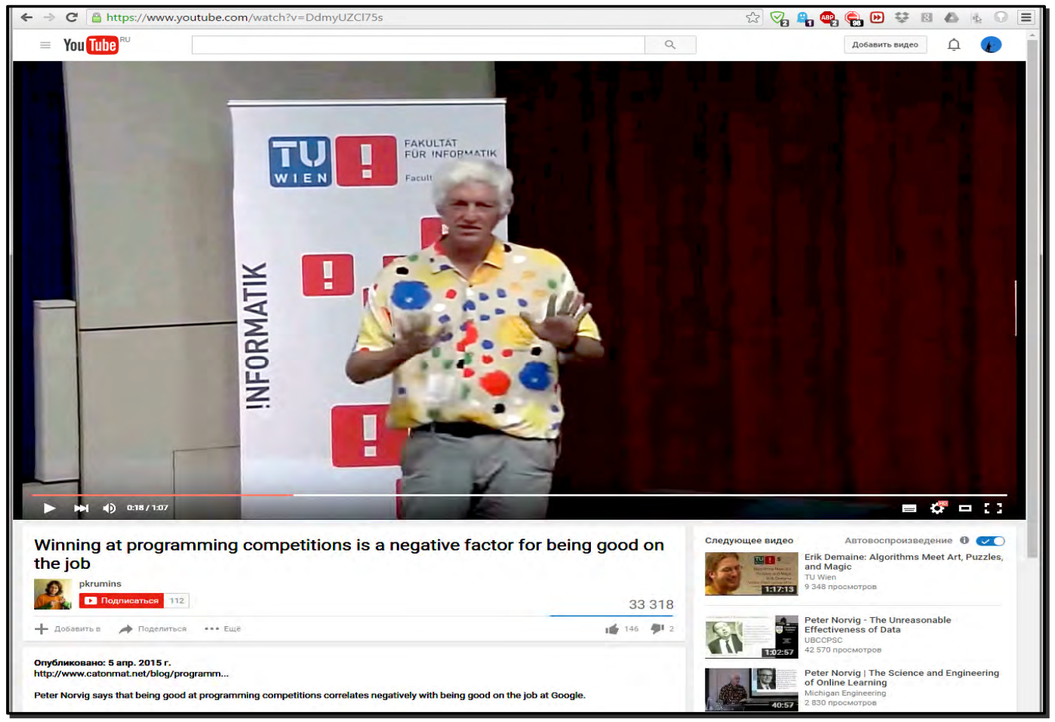

Олимпиадный путь  Нереальный путь  Проектный путь

# Модели опыта – профессионального и контестного




| | | |
|---|---|---|
| Локальную абстрактную задачу | Ориентация на | Целостный продукт |
| Небольшой | Объем задачи | Большой |
| Узкий и постоянный | Контекст задачи | Широкий и меняющийся |
| Импульсная фрагментарная | Характер работы | Непрерывная итерационная |
| Только со своим | Работа с кодом | Со своим и чужим |
| Индивидуальная | Разработка | Командная |
| Мало | Стандарты и литература | Много |
| Нет | Документация и сопровождение | Обязательны |

# Модели опыта – профессионального и контестного




**«Это было удивительно для меня, но победы в конкурсах по программированию негативно коррелируют с успехами в работе»**

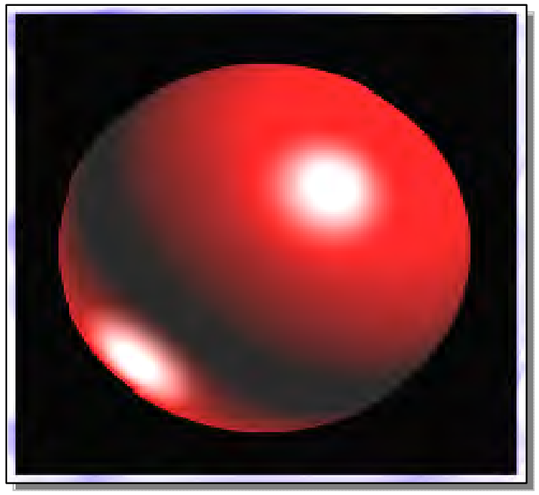
**Питер Норвиг (Peter Norvig), директор по исследованиям Google, советник Ассоциации по улучшению искусственного интеллекта, автор одного из самых популярных вузовских учебников по ИИ**

| | | |
|---|---|---|
| Локальную абстрактную задачу | Ориентация на | Целостный продукт |
| Небольшой | Объем задачи | Большой |
| Узкий и постоянный | Контекст задачи | Широкий и меняющийся |
| Импульсная фрагментарная | Характер работы | Непрерывная итерационная |
| Только со своим | Работа с кодом | Со своим и чужим |
| Индивидуальная | Разработка | Командная |
| Мало | Стандарты и литература | Много |
| Нет | Документация и сопровождение | Обязательны |

# Что такое культура разработки?

**Система ценностей и критериев качества**

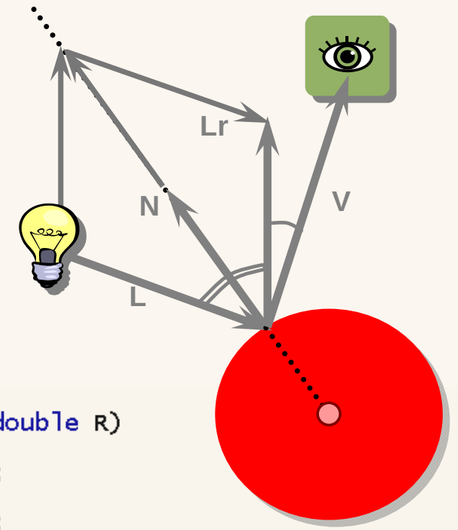
- Ясность
  Выразительность
- Надежность
  Сопровождаемость
- Модульность
  Архитектурная логичность
- Масштабируемость
- Поддержка стандартов
  Переносимость
- Навыки командной работы
- **Культура как контекст**



```
int main()
{
double t,j,i,px,py,pz,q,nx,ny,nz,vx,vy,vz,lx,ly,lz;
double diff,lpx,lpy,lpz,vpx,vpy,vpzr,g,b;
vpx=vpy=0;
vpz=500;
for(t=3.14/2;;t+=0.1)
for(j=-100;j<=100;j++)
for (i=-100;i<=100;i++) {
if (i*i+j*j>10000) continu
lpx=-200*(1+cos(t));
lpy=-200*cos(t);
lpz=200*sin(t);
px=i; py=j; pz=sqrt(10000-i*i-j*j));
q=sqrt(px*px+py*py+pz*pz);
nx=px/q,ny=py/q,nz=pz/q;
q=sqrt((vpx-px)*(vpx-px)+(wpy-py)*(wpy-py)+(wpz-pz)*(wpz-pz))
vx=(wpx-px)/q; vy=(wpy-py)/q;vz=(wpz-pz)/q;
q=sqrt((lpx-px)*(lpx-px)+(lpy-py)*(lpy-py)+(lpz-pz)*(lpz-pz))
vx=(lpx-px)/q; vy=(lpy-py)/q; vz=(lpz-pz)/q;
diff=vx*lx+vy*ly+vz*lz; if(diff<0) diff=0;
r=b=0.2;
g=0.07*diff+0.2;
if(r>1)r=1; if(r<0)r=0; if(g>1)g=1; if(g<0)g=0; if(b>1)b=1; i
SetPixel(hwnd,i+100,j+100,RGB(r*255,g*255,b*255));
}
}
```



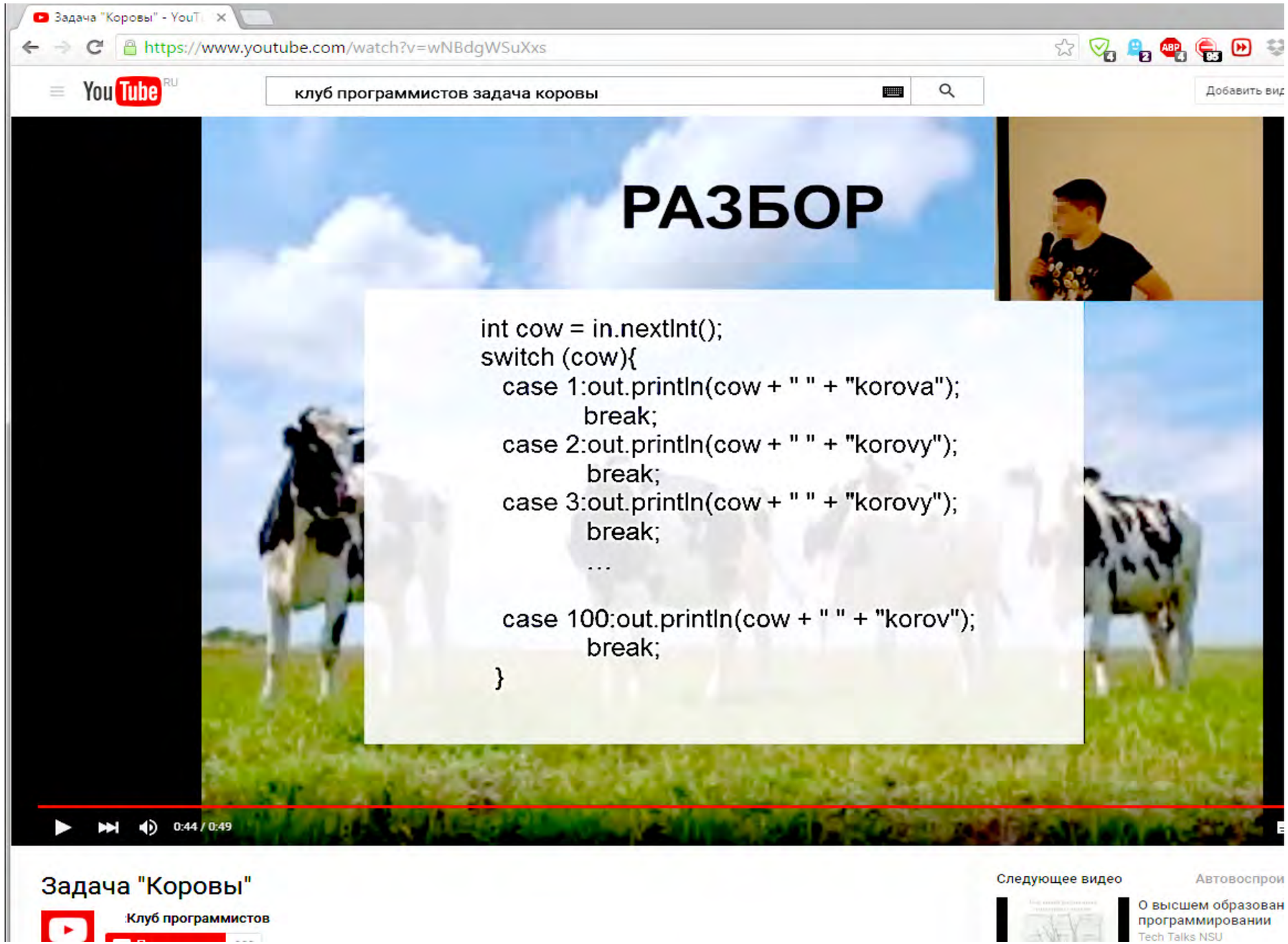
```
//--------------------------------

int main()
    {
    double R = 100;

    txCreateWindow (2*R, 2*R);

    for (double t = txPI/2; ; t +=
        DrawScene (vec (-2*R * (1 +
                        -2*R * cos
                        +2*R * sin
    return 0;
    }

//--------------------------------

void DrawScene (const vec& lightPos, double R)
    {
    vec viewPos       (  0,  0, +5*R);

    vec materialColor (0.0, 1.0, 0.0);
    vec lightColor    (1.0, 0.7, 0.0);
    vec ambientColor  (0.2, 0.2, 0.2);

    for (double y = -R; y <= R; y++)
        for (double x = -R; x <= R; x++)
            {
            if (x*x + y*y > R*R) continue;

            vec p (x, y, sqrt (R*R - x*x - y*y));
            vec N = !p;

            vec V = ! (viewPos  - p);
            vec L = ! (lightPos - p);

            double diffuse = N ^ L;
            if (diffuse < 0) diffuse = 0;

            vec Lr = 2*(N^L)*N - L;
            double spec = Lr ^ V;
            if (spec < 0) spec = 0;
            double specular = pow (spec, 25);

            vec color = ambientColor * materialColor +
                        diffuse      * materialColor * lightColor +
                        specular     * lightColor);

            DrawPixel (x+R, y+R, color);
            }
    }
```



# Что такое культура разработки?

**Система ценностей и критериев качества**

- Ясность
  Выразительность
- Надежность
  Сопровождаемость
- Модульность
  Архитектурная логичность
- Масштабируемость
- Поддержка стандартов
  Переносимость
- Навыки командной работы
- **Культура как контекст**



# Основные проблемы в разработке

- **Проблема большого кода**
  - Много связей
  - Вы теряете власть над кодом (управляемость)
  - Странные ошибки, нестыковки
  - «Костыли»
- **Неясный код**
  - Коллеги ничего не понимают
  - Вы сами ничего не понимаете спустя полгода
- **Ненадежный код**
  - Не проверяются ошибки

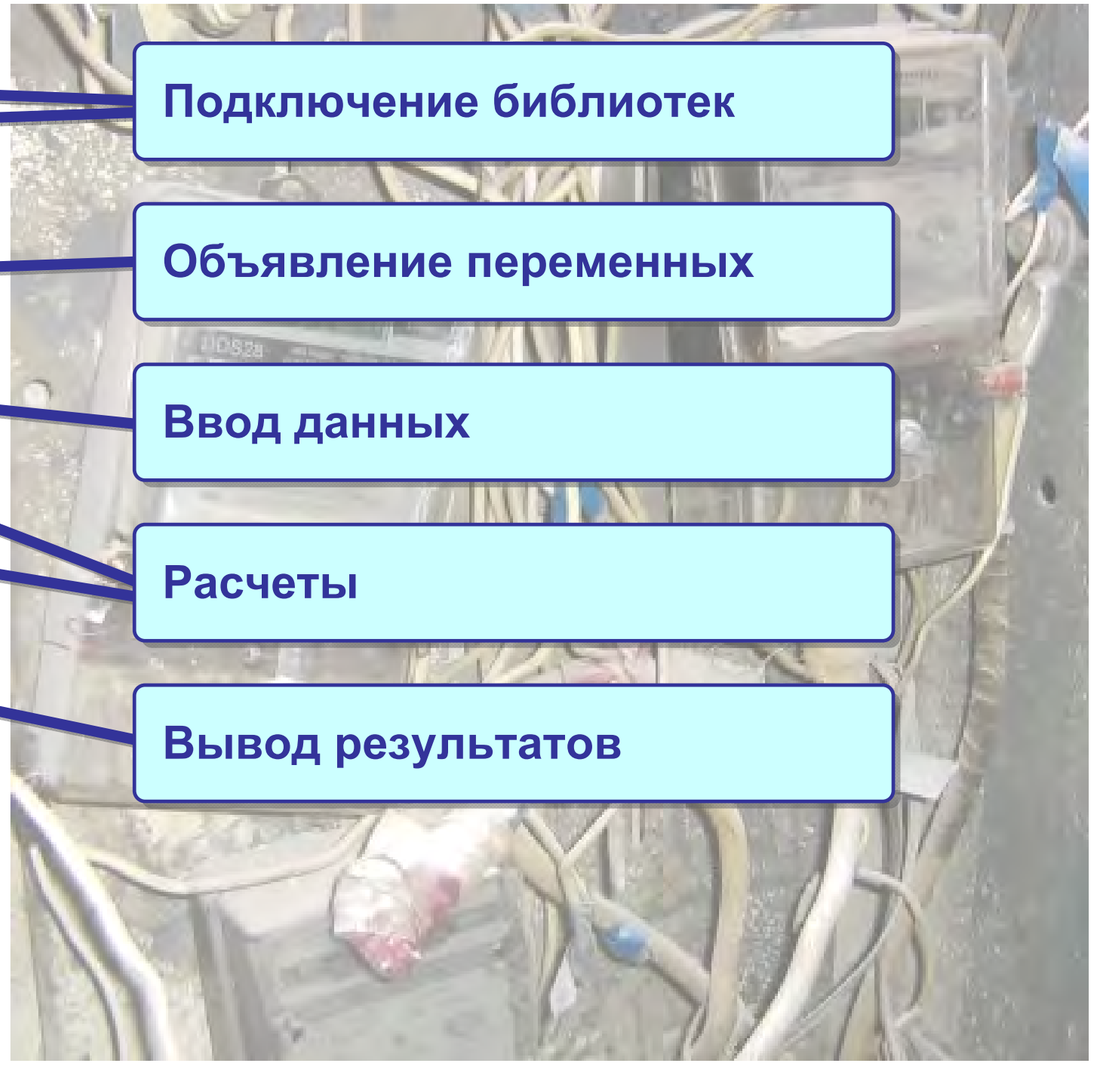
**А в результате падают спутники и ракеты**

# Квадратное уравнение бросает вызов

```
#include<stdio.h>
#include   <math.h>
int main ( ){
  float a,b,c,p,q;
 scanf(“%g %g %g”,&a,&b,&c);
  p= (-b  + sqrt (b*b –4*a*c))/2*a;
 q=(-b + sqrt (b*b–4*a*c)) /2/a;
printf(“%g %g”,p,q);}
```

# Квадратное уравнение бросает вызов

```
#include<stdio.h>
#include   <math.h>
int main ( ){
  float a,b,c,p,q;
 scanf(“%g %g %g”,&a,&b,&c);
  p= (-b  + sqrt (b*b -4*a*c))/2*a;
q=(-b + sqrt (b*b-4*a*c)) /2/a;
printf(“%g %g”,p,q);}
```



Подключение библиотек

Объявление переменных

Ввод данных

Расчеты

Вывод результатов

# Квадратное уравнение бросает вызов

```
#include<stdio.h>
#include   <math.h>
int main ( ){
  float a,b,c,p,q;
 scanf(“%g %g %g”,&a,&b,&c);
  p= (-b  + sqrt (b*b –4*a*c))/2*a;
q=(-b + sqrt (b*b–4*a*c)) /2/a;
printf(“%g %g”,p,q);}
```

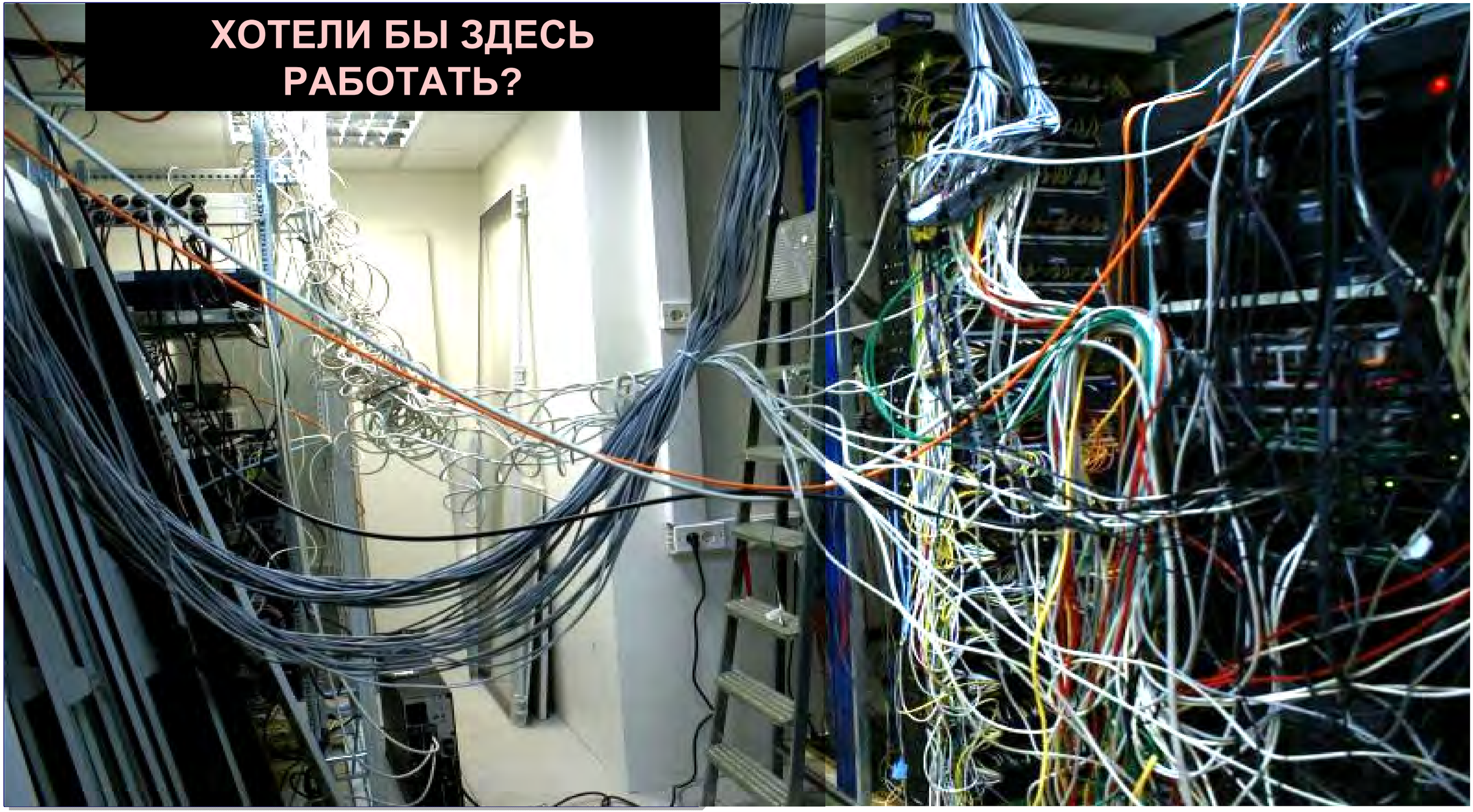
**FFFFUUUUUUUUU!**

**Не надо так!**

# Квадратное уравнение бросает вызов

```
// Jetto drugaya zaddatschzhschua
#include<stdio.h>
#include   <math.h>
#include <stdlib.h>
int main(){
  float a,b,c,p,q,s;
for(int i=1;i<=100;i++){
 a=rand()%2001-1000;
 b=rand()%2001-1000;
 c=rand()%2001-1000;
p= (-b  + sqrt (b*b –4*a*c))/2*a;
 q=(-b + sqrt (b*   b–4*a*c)) /2/a;
  s=s+p+q;
} printf   (“%g”,s);}
}
```

# Квадратное уравнение бросает вызов



# Квадратное уравнение бросает вызов

```
#include<stdio.h>
#include   <math.h>
#include <stdlib.h>
int main(){
  float a,b,c,p,q,s;
for(int i=1;i<=100;i++){
 a=rand()%2001-1000;
 a=rand()%2001-1000;
 c=rand()%2001-1000;
p==(-b  + sqrt (b*b –4*a*c))/2*a;
 q =(-b+ sqrt (b*   b–4*a*c)) /2/a;
  s=s+p+q;
} printf   (“%g”,s);}
}
```

**Проблемы кода:**

- **Код не читается (плохой стиль)**
- **Неясные имена переменных**
  **Нет инициализации переменных**
- **Слишком формальный подход (a == 0 не рассматривается)**
- **Логические ошибки**
- **Повторные вычисления**
- **Неясный ввод-вывод**
- **Монолитный код (две задачи слиты вместе)**

  **Нельзя использовать в других задачах как часть решения**

# Квадратное уравнение бросает вызов

## Что же делать???

# Квадратное уравнение бросает вызов

```
int main()
  {

  int nRoots = SolveSquare (a, b, c, &x1, &x2);

return 0;
}
```

- **«Разделяй и властвуй»**
  **(с) первый программист в истории – Юлий Цезарь**
  - В любой непонятной ситуации делай новую функцию :)

# Квадратное уравнение бросает вызов

```
int main()
  {
  printf (“# Square equation solver\n”
          “# (c) Ded, 2017\n\n”);

  printf (“# Enter a, b, c: ”);

  double a = 0, b = 0, c = 0;
  scanf (“%lg %lg %lg”, &a, &b, &c);

  double x1 = 0, x2 = 0;
  int nRoots = SolveSquare (a, b, c, &x1, &x2);

  switch (nRoots)
    {
    case 0:  printf (“No roots\n”);
             break;

    case 1:  printf (“x = %lg\n”, x1);
             break;

    case 2:  printf (“x1 = %lg, x2 = %lg\n”, x1, x2);
             break;

    case SS_INF_ROOTS: printf (“Any number”);
             break;

    }

return 0;
}
```

- **«Разделяй и властвуй»**
  **(c) первый программист в истории – Юлий Цезарь**
  - В любой непонятной ситуации делай новую функцию :)
- **Разработка «сверху вниз»**
  - Пишем сначала главную программу
  - Считаем, что «все уже написано»
  - Легкость и свобода
  - Можно легко менять вызовы функций, как удобнее – ведь они еще не написаны
  - После этого пишем сами функции, при необходимости повторяя процесс

# Квадратное уравнение бросает вызов

```
int main()
  {
  printf (“# Square equation solver\n”
          “# (c) Ded, 2017\n\n”);

  printf (“# Enter a, b, c: ”);

  double a = 0, b = 0, c = 0;
  scanf (“%lg %lg %lg”, &a, &b, &c);

  double x1 = 0, x2 = 0;
  int nRoots = SolveSquare (a, b, c, &x1, &x2);

  switch (nRoots)
    {
    case 0:  printf ("No roots\n");
             break;

    case 1:  printf ("x = %lg\n", x1);
             break;

    case 2:  printf ("x1 = %lg, x2 = %lg\n", x1, x2);
             break;

    case SS_INF_ROOTS: printf ("Any number");
             break;

    }

return 0;
}
```

- **«Разделяй и властвуй»**
  **(с) первый программист в истории – Юлий Цезарь**
  - В любой непонятной ситуации делай новую функцию :)
- **Разработка «сверху вниз»**
  - Пишем сначала главную программу
  - Считаем, что «все уже написано»
  - Легкость и свобода
  - Можно легко менять вызовы функций, как удобнее – ведь они еще не написаны
  - После этого пишем сами функции, при необходимости повторяя процесс

# Квадратное уравнение бросает вызов

```
int main()
  {
  printf (“# Square equation solver\n”
          “# (c) Ded, 2017\n\n”);

  printf (“# Enter a, b, c: ”);

  double a = 0, b = 0, c = 0;
  scanf (“%lg %lg %lg”, &a, &b, &c);

  double x1 = 0, x2 = 0;
  int nRoots = SolveSquare (a, b, c, &x1, &x2);

  switch (nRoots)
    {
    case 0:  printf (“No roots\n”);
             break;

    case 1:  printf (“x = %lg\n”, x1);
             break;

    case 2:  printf (“x1 = %lg, x2 = %lg\n”, x1, x2);
             break;

    case SS_INF_ROOTS: printf (“Any number”);
             break;

    }

return 0;
}
```

- **«Разделяй и властвуй»**
  **(с) первый программист в истории – Юлий Цезарь**
  - В любой непонятной ситуации делай новую функцию :)
- **Разработка «сверху вниз»**
  - Пишем сначала главную программу
  - Считаем, что «все уже написано»
  - Легкость и свобода
  - Можно легко менять вызовы функций, как удобнее – ведь они еще не написаны
  - После этого пишем сами функции, при необходимости повторяя процесс

# Квадратное уравнение бросает вызов

```
int main()
  {
  printf (“# Square equation solver\n”
          “# (c) Ded, 2017\n\n”);

  printf (“# Enter a, b, c: ”);

  double a = 0, b = 0, c = 0;
  scanf (“%lg %lg %lg”, &a, &b, &c);

  double x1 = 0, x2 = 0;
  int nRoots = SolveSquare (a, b, c, &x1, &x2);

  switch (nRoots)
    {
    case 0:  printf (“No roots\n”);
             break;

    case 1:  printf (“x = %lg\n”, x1);
             break;

    case 2:  printf (“x1 = %lg, x2 = %lg\n”, x1, x2);
             break;

    case SS_INF_ROOTS: printf (“Any number”);
             break;
             return 1;
    }

return 0;
}
```

- **«Разделяй и властвуй»**
  **(с) первый программист в истории – Юлий Цезарь**
  - В любой непонятной ситуации делай новую функцию :)
- **Разработка «сверху вниз»**
  - Пишем сначала главную программу
  - Считаем, что «все уже написано»
  - Легкость и свобода
  - Можно легко менять вызовы функций, как удобнее – ведь они еще не написаны
  - После этого пишем сами функции, при необходимости повторяя процесс

# Квадратное уравнение бросает вызов

```
int main()
  {
  printf (“# Square equation solver\n”
          “# (c) Ded, 2017\n\n”);

  printf (“# Enter a, b, c: ”);

  double a = 0, b = 0, c = 0;
  scanf (“%lg %lg %lg”, &a, &b, &c);

  double x1 = 0, x2 = 0;
  int nRoots = SolveSquare (a, b, c, &x1, &x2);

  switch (nRoots)
    {
    case 0:  printf (“No roots\n”);
             break;

    case 1:  printf (“x = %lg\n”, x1);
             break;

    case 2:  printf (“x1 = %lg, x2 = %lg\n”, x1, x2);
             break;

    case SS_INF_ROOTS: printf (“Any number”);
             break;

    default: printf (“main(): ERROR: nRoots = %d\n”,
                     nRoots);
             return 1;
    }
```

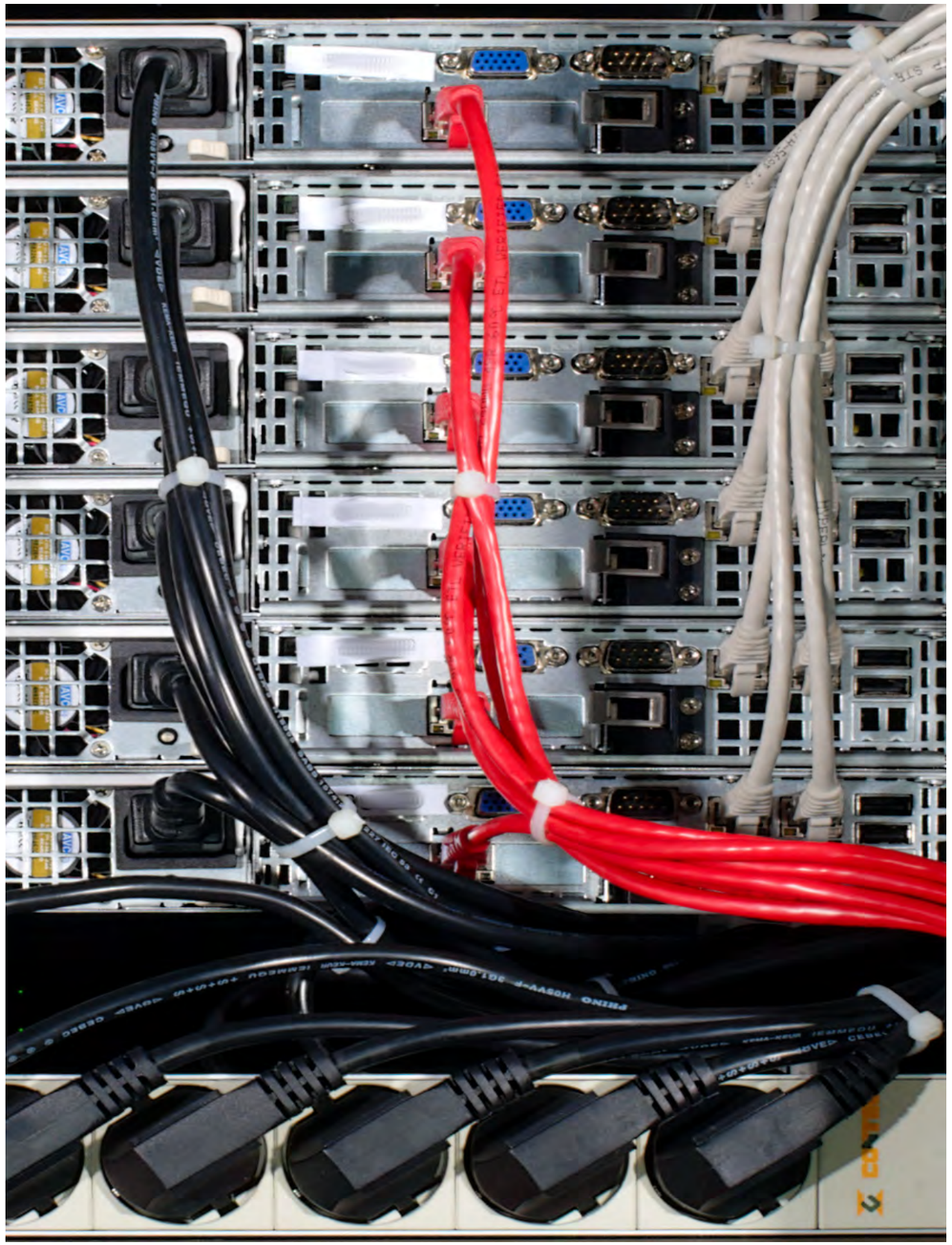
- **«Разделяй и властвуй»**
  **(с) первый программист в истории – Юлий Цезарь**
  - В любой непонятной ситуации делай новую функцию :)
- **Разработка «сверху вниз»**
  - Пишем сначала главную программу
  - Считаем, что «все уже написано»
  - Легкость и свобода
  - Можно легко менять вызовы функций, как удобнее – ведь они еще не написаны
  - После этого пишем сами функции, при необходимости повторяя процесс

# Квадратное уравнение бросает вызов

```
int main()
  {
  printf (“# Square equation solver\n”
          “# (c) Ded, 2017\n\n”);

  printf (“# Enter a, b, c: ”);

  double a = 0, b = 0, c = 0;
  scanf (“%lg %lg %lg”, &a, &b, &c);

  double x1 = 0, x2 = 0;
  int nRoots = SolveSquare (a, b, c, &x1, &x2);

  switch (nRoots)
    {
    case 0:  printf (“No roots\n”);
             break;

    case 1:  printf (“x = %lg\n”, x1);
             break;

    case 2:  printf (“x1 = %lg, x2 = %lg\n”, x1, x2);
             break;

    case SS_INF_ROOTS: printf (“Any number”);
             break;

    default: printf (“main(): ERROR: nRoots = %d\n”,
                     nRoots);
             return 1;
    }
```

# Квадратное уравнение бросает вызов

```
int SolveSquare (double a, double b, double c,
                 double* x1, double* x2)
  {
  if (a == 0)
    {
    if (b == 0)
      {
      return (c == 0)? SS_INF_ROOTS : 0;
      }
    else /* if (b != 0) */
      {
      *x1 = -c / b;
      return 1;
      }
    }

  else /* if (a != 0) */
    {
    double d = b*b - 4*a*c;

    if (d == 0)
      {
      *x1 = *x2 = -b / (2*a);
      return 1;
      }
    else
      {
      double sqrt_d = sqrt (d);

      *x1 = (-b - sqrt_d) / (2*a);
      *x2 = (-b + sqrt_d) / (2*a);

      return 2;
```



# Квадратное уравнение уже не бросает вызов

```
int SolveSquare (double a, double b, double c,
                 double* x1, double* x2)
  {
  assert (std::isfinite (a));
  assert (std::isfinite (b));
  assert (std::isfinite (c));

  assert (x1 != NULL);
  assert (x2 != NULL);
  assert (x1 != x2);

  if (a == 0)
    {
    if (b == 0)
      {
      return (c == 0)? SS_INF_ROOTS : 0;
      }
    else /* if (b != 0) */
      {
      *x1 = -c / b;
      return 1;
      }
    }

  else /* if (a != 0) */
    {
    double d = b*b - 4*a*c;

    if (d == 0)
      {
      *x1 = *x2 = -b / (2*a);
      return 1;
      }
```

- **Добавляем проверки входных параметров**
- **assert() при появлении ложного утверждения прерывает программу и выдает диагностику, где это случилось:**

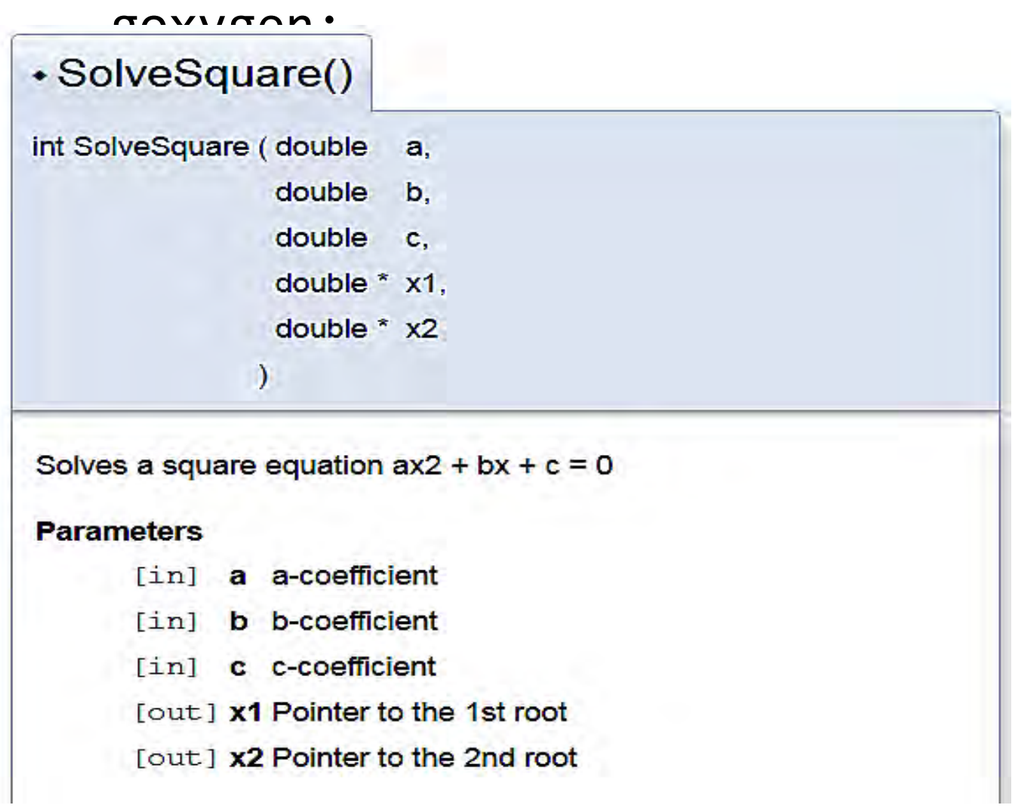
```
Assertion failed: x1 != x2,
file: SolveSquare_v2.cpp,
line: 42
```

**Сразу ясно, где сломалось**

# Квадратное уравнение уже не бросает вызов

```
//-----------------------------------------------
//! Solves a square equation ax2 + bx + c = 0
//!
//! @param [in]  a   a-coefficient
//! @param [in]  b   b-coefficient
//! @param [in]  c   c-coefficient
//! @param [out] x1  Pointer to the 1st root
//! @param [out] x2  Pointer to the 2nd root
//!
//! @return Number of roots
//!
//! @note   In case of infinite number of roots,
//!         returns SS_INF_ROOTS.
//-----------------------------------------------

int SolveSquare (double a, double b, double c,
                 double* x1, double* x2)
  {
  ...

  assert (x1 != NULL);
  assert (x2 != NULL);
  assert (x1 != x2);

  if (a == 0)
    {
    if (b == 0)
      {
      return (c == 0)? SSQ_INF_ROOTS : 0;
      }
    else /* if (b != 0) */
      {
      *x1 = -c / b;
      return 1;
```

- **Добавляем документацию**
  - Теперь нас не будут дергать из отпуска за наш счет, если какая-то часть программы неясна коллегам
  - Вот как это выглядит при обработке системой автоматической документации doxygen:



• SolveSquare()

int SolveSquare ( double a,
double b,
double c,
double * x1,
double * x2
)

Solves a square equation ax2 + bx + c = 0

**Parameters**

[in] **a** a-coefficient
[in] **b** b-coefficient
[in] **c** c-coefficient
[out] **x1** Pointer to the 1st root
[out] **x2** Pointer to the 2nd root

# Квадратное уравнение уже не бросает вызов

## • SolveSquare()

```
int SolveSquare ( double    a,
                  double    b,
                  double    c,
                  double *  x1,
                  double *  x2
                )
```

Solves a square equation ax2 + bx + c = 0

**Parameters**

[in] **a** a-coefficient

[in] **b** b-coefficient

[in] **c** c-coefficient

[out] **x1** Pointer to the 1st root

[out] **x2** Pointer to the 2nd root

**Returns**

Number of roots

**Note**

In case of infinite number of roots, returns SS_INF_ROOTS.

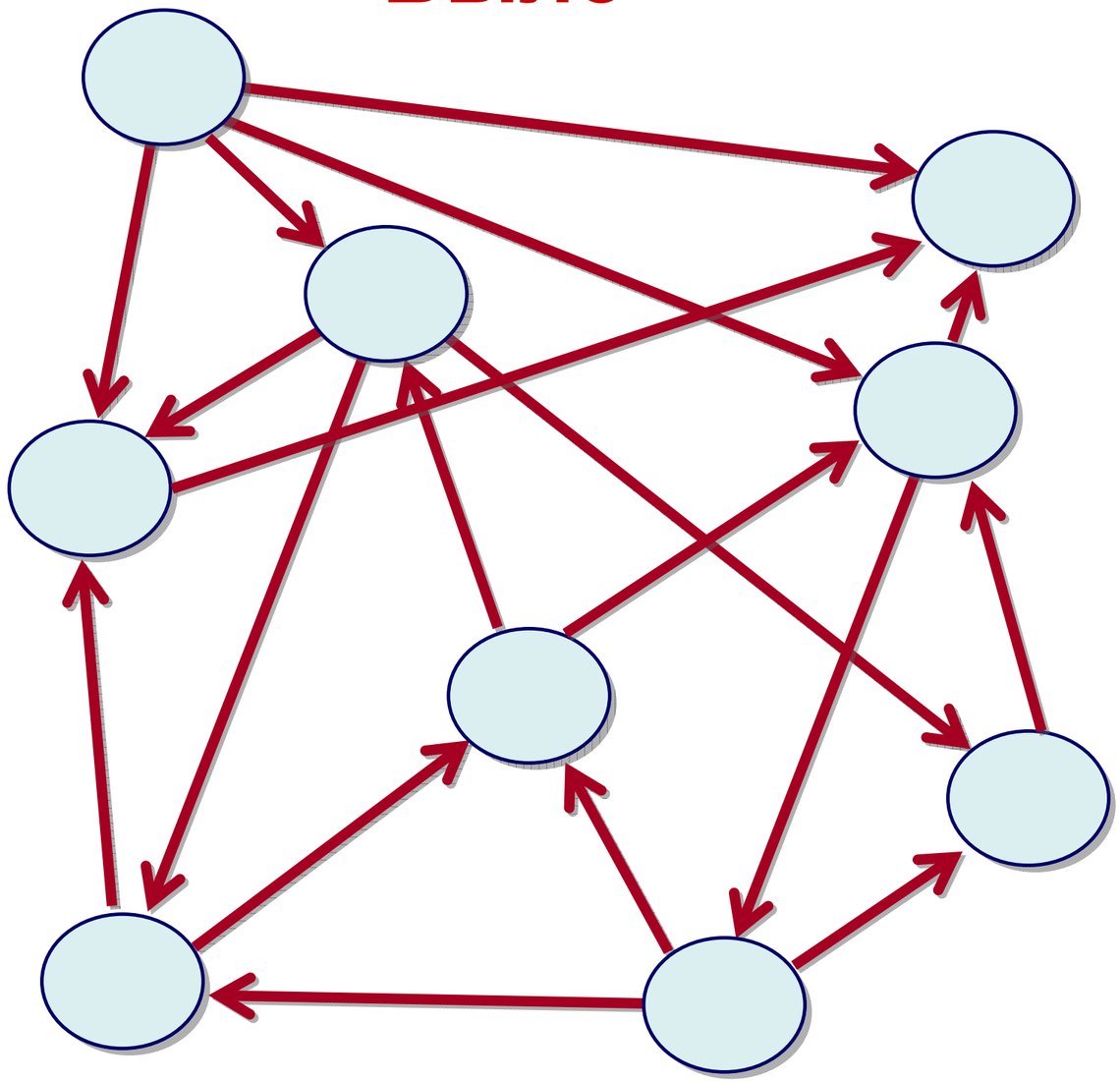
Referenced by **main()**.

```
19      {
20      assert (x1 != NULL);
21      assert (x2 != NULL);
22      assert (x1 != x2);
23
24      if (a == 0)
25          {
26          if (b == 0)
27              {
28              return (c == 0)? SS_INF_ROOTS : 0;
29              }
30          else /* if (b != 0) */
31              {
32              *x1 = -c / b;
33              return 1;
34              }
35          }
36
37      else /* if (a != 0) */
38          {
39          double d = b*b - 4*a*c;
40
41          if (d == 0)
42              {
43              *x1 = *x2 = -b / (2*a);
44              return 1;
45              }
46          else
47              {
48              double sqrt_d = sqrt (d);
49
50              *x1 = (-b - sqrt_d) / (2*a);
51              *x2 = (-b + sqrt_d) / (2*a);
52
53              return 2;
54              }
55          }
56      }
```

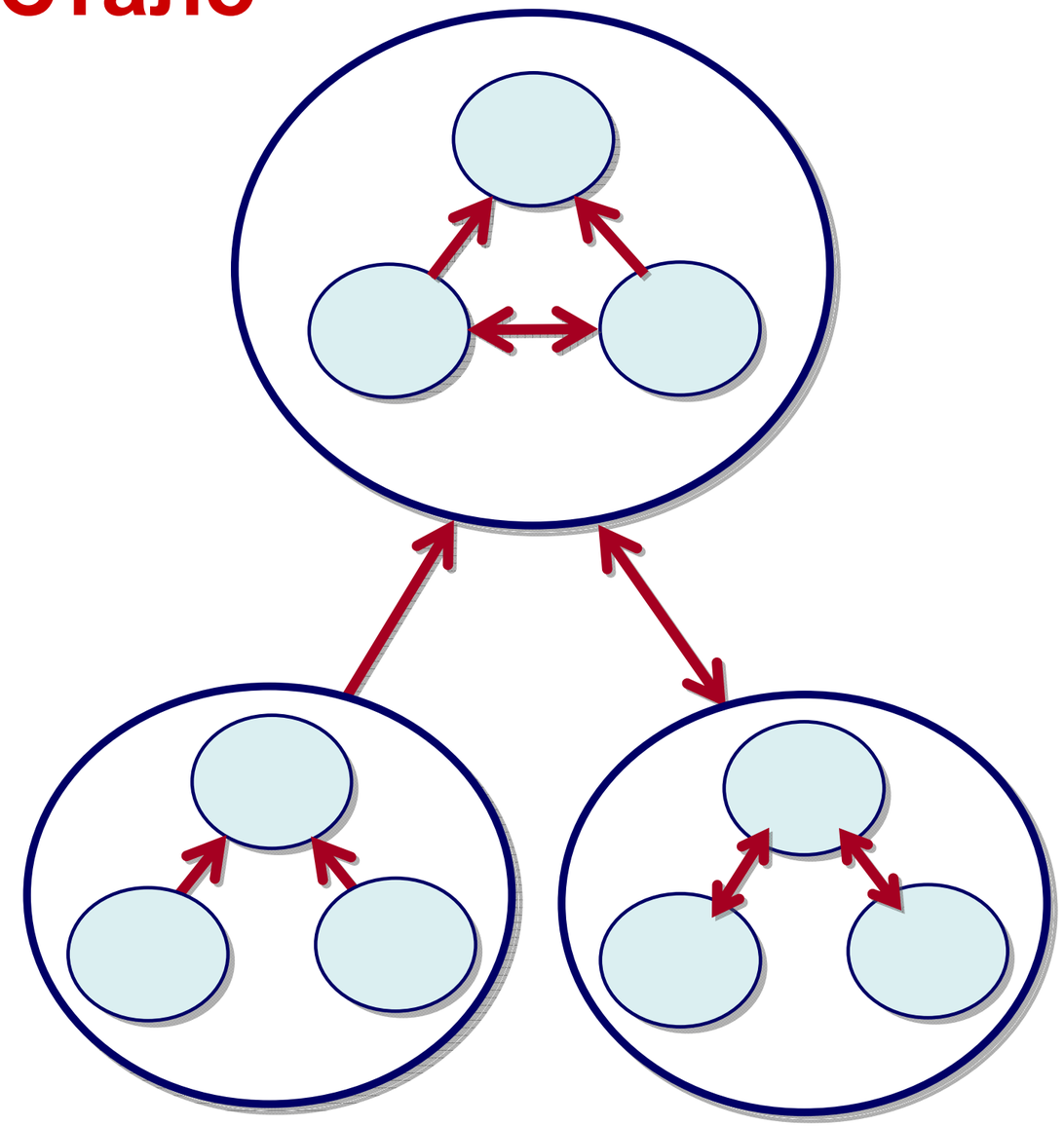
Here is the caller graph for this function:

# Что получаем?

## Было



## Стало

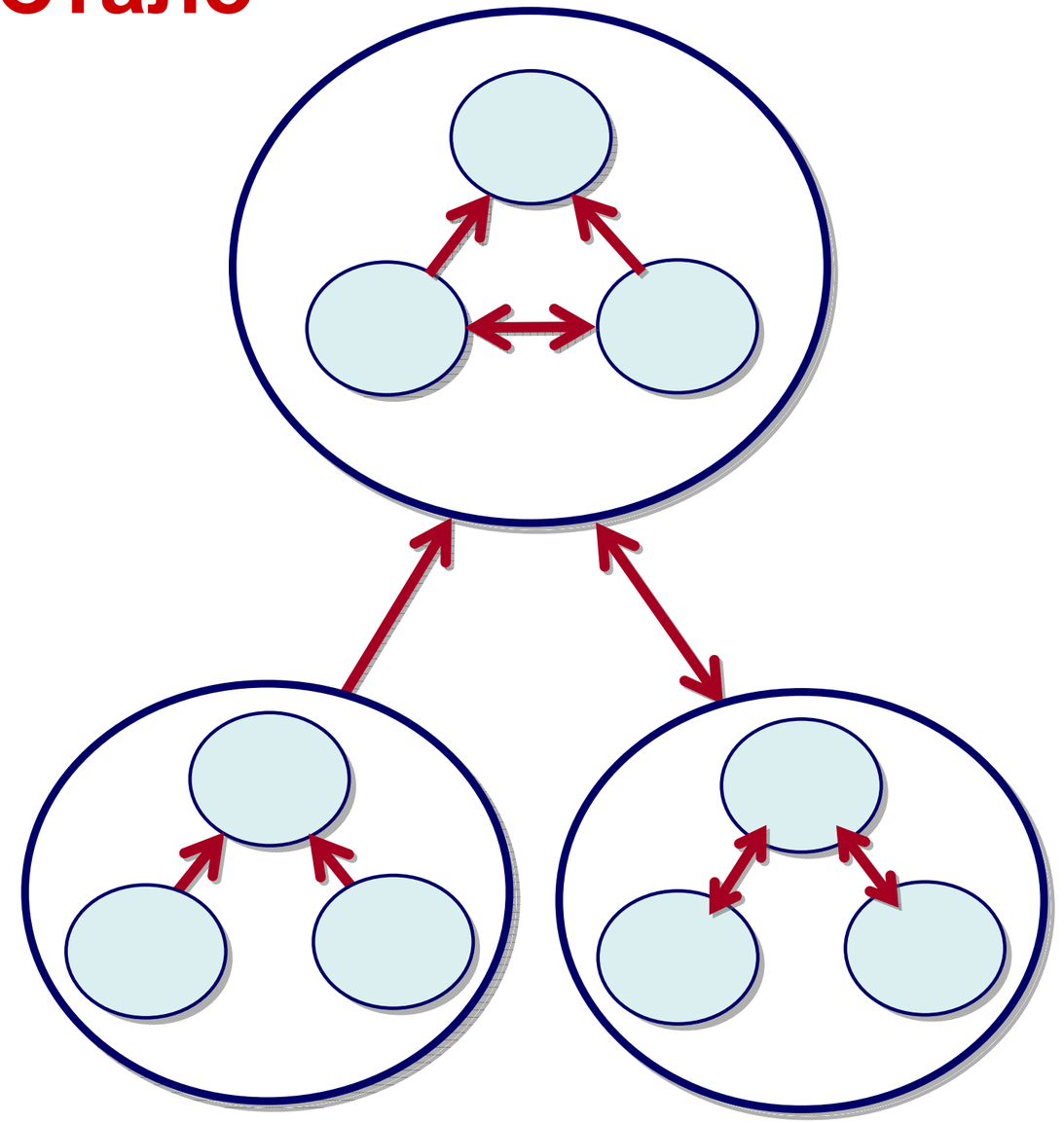


# Что получаем?

- Легко поддерживать
- Легко изменять
- Легко тестировать
- Легко распределять по членам команды
- Легко использовать отдельные модули в других проектах

…Чтобы не было мучительно стыдно за бесцельно написанный код.

Стало



# Промышленное программирование

- **Учебная группа на первом курсе ФРТК**

- **Доп. курс «Введение в промышленное программирование» для ФРТК**
- **Курс промышленного программирования на «Технотреке» Mail.Ru (все факультеты)**

**Тематика занятий:**

- **Системное программирование**
- **Моделирование вычислительных систем**
- **Разработка компиляторов собственных языков программирования**

**Спасибо за внимание!**

**Регистрация на школу программирования для абитурентов МФТИ**
**5-12 июля, 12:20-16:00, 7 этаж КПМ**

**http://gg.gg/MIPT-Summer-2017**

**(Предполагается уже имеющийся олимпиадный или проектный опыт!)**

**Задавайте вопросы!**

# Книжки

## C++

- Б. Страуструп, «Программирование. Принципы и практика с использованием C++» (2-е издание)
- Б. Страуструп, «Дизайн и эволюция языка C++»
- С. Прата, «Язык C++ для начинающих»
- Дж. Лафоре, «Объектно-ориентированное программирование в C++»

# Книжки

**Более глубокое**

- С. Мейерс, «Эффективное использование C++» (все 3 книги)
- Г. Саттер, А. Александреску, «Стандарты программирования на C++»
- Г. Саттер, «Решение сложных задач на C++», «Новые сложные задачи на C++»
- Дж. Элджер, «Эффективный C++»
- Д. Вандевурд, Н. Джосаттис, «Шаблоны C++. Справочник разработчика»
- А. Александреску, «Современное проектирование на C++»
- Дж. Коплиен «Мультипарадигменное проектирование на C++»

# Книжки

## Общие вещи

- С. МакКоннелл, «Совершенный код»
- М. Фаулер, «Рефакторинг»
- Ю. Хсих, «Наука отладки»

## Паттерны проектирования

- Дж. Влиссидес, «Применение шаблонов проектирования»
- Э. Гамма, Г. Хелм, Г. Джонсон, Дж. Влиссидес, «Приемы объектно-ориентированного проектирования. Паттерны проектирования»

## Философия :)

- Р. Брукс, «Мифический человеко-месяц»
- Э. Хант, Д. Томас, «Программист-прагматик»