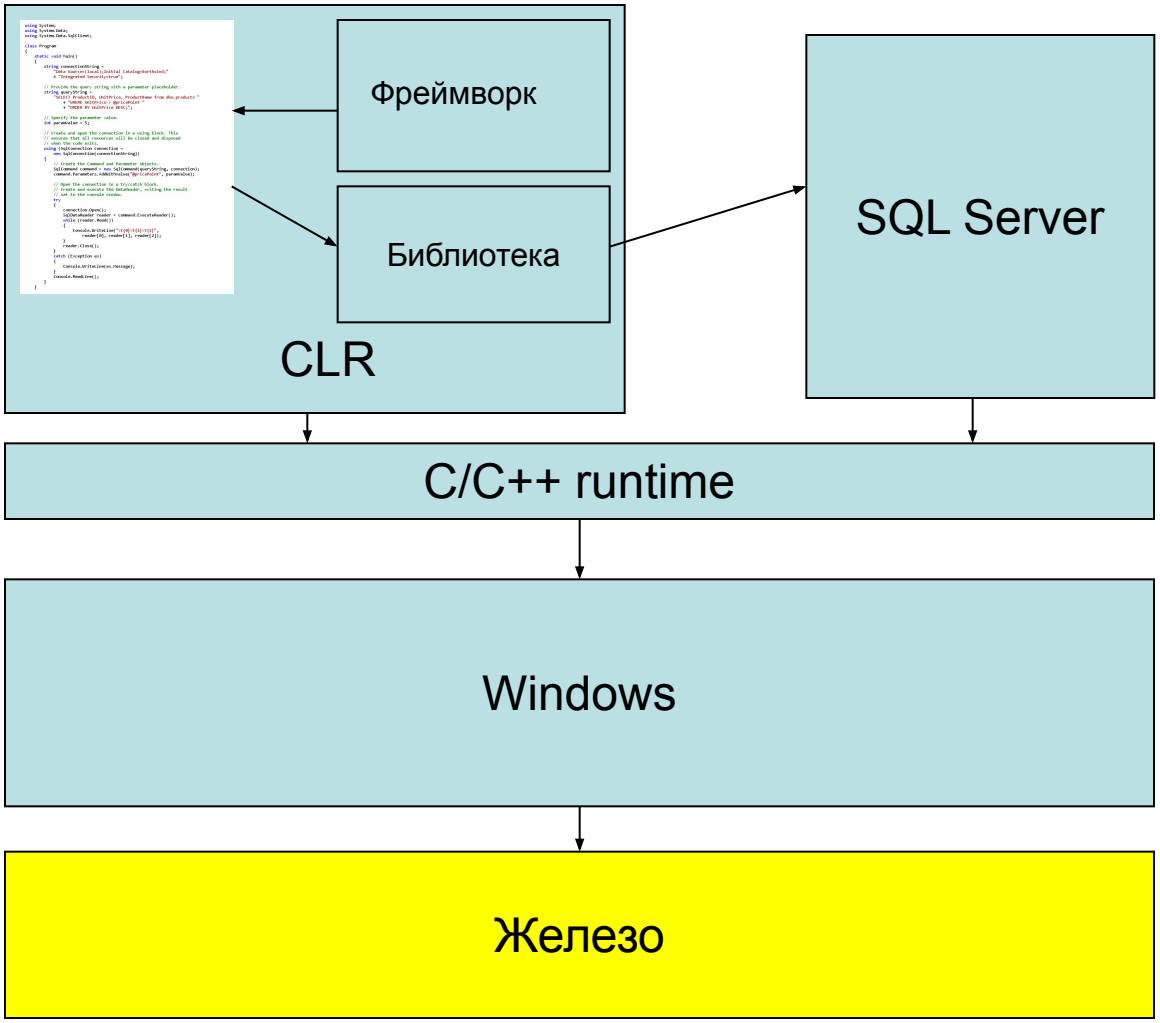
Протекающие абстракции

Или
зачем современному программисту
знать все эти низкоуровневые
детали

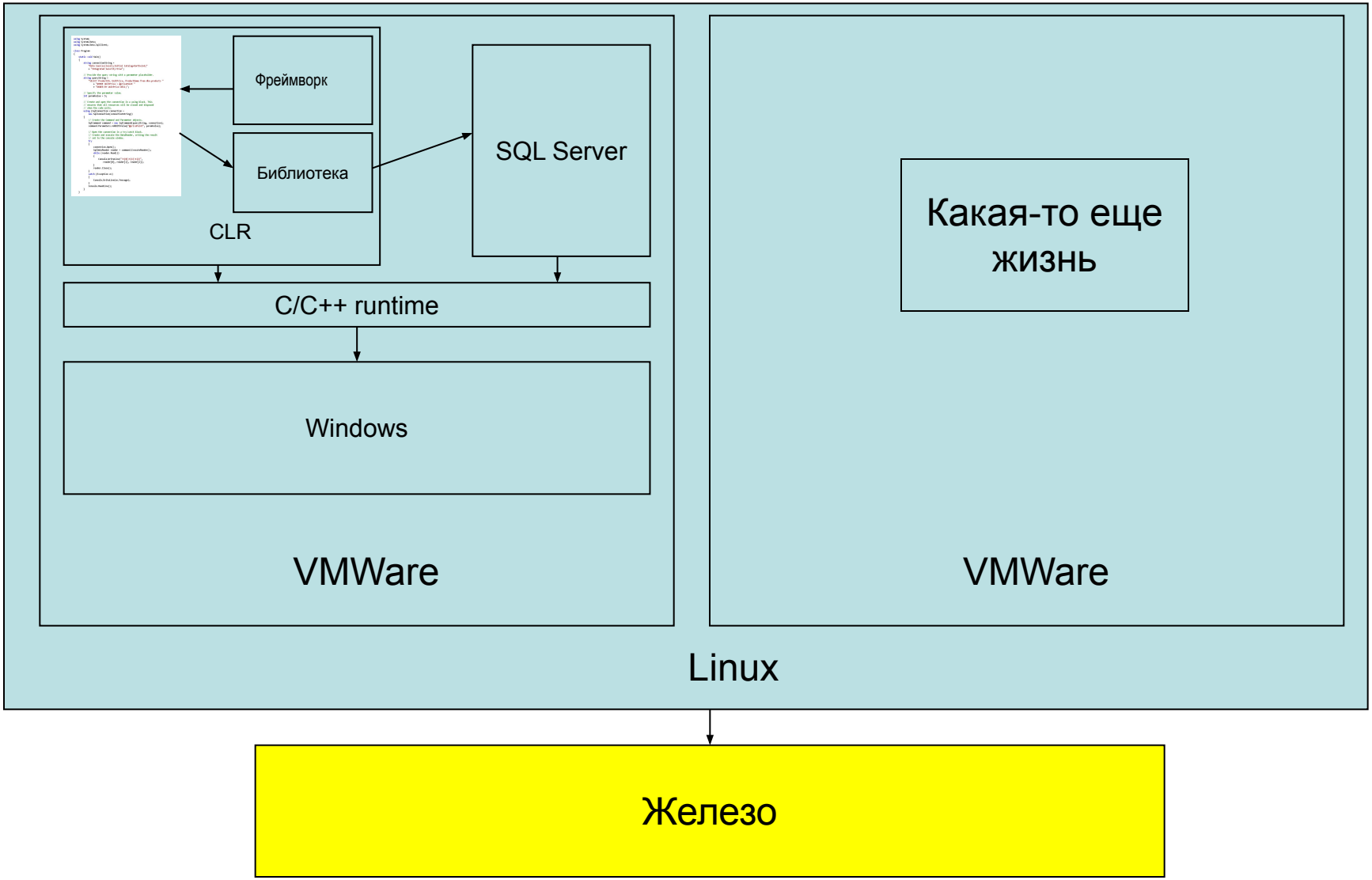
Что пишет типичный современный программист

```
using System;
using System.Data;
using System.Data.SqlClient;

class Program
{
    static void Main()
    {
        string connectionString =
            "Data Source=(local);Initial Catalog=Northwind;"
            + "Integrated Security=true";

        // Provide the query string with a parameter placeholder.
        string queryString =
            "SELECT ProductID, UnitPrice, ProductName from dbo.products "
                + "WHERE UnitPrice > @pricePoint "
                + "ORDER BY UnitPrice DESC;";

        // Specify the parameter value.
        int paramValue = 5;

        // Create and open the connection in a using block. This
        // ensures that all resources will be closed and disposed
        // when the code exits.
        using (SqlConnection connection =
            new SqlConnection(connectionString))
        {
            // Create the Command and Parameter objects.
            SqlCommand command = new SqlCommand(queryString, connection);
            command.Parameters.AddWithValue("@pricePoint", paramValue);

            // Open the connection in a try/catch block.
            // Create and execute the DataReader, writing the result
            // set to the console window.
            try
            {
                connection.Open();
                SqlDataReader reader = command.ExecuteReader();
                while (reader.Read())
                {
                    Console.WriteLine("\t{0}\t{1}\t{2}",
                        reader[0], reader[1], reader[2]);
                }
                reader.Close();
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
            Console.ReadLine();
        }
    }
```

Что пишет типичный современный программист



А то и еще хуже



На самом деле все еще хуже

Препроцессор
шаблоны
условная компиляция

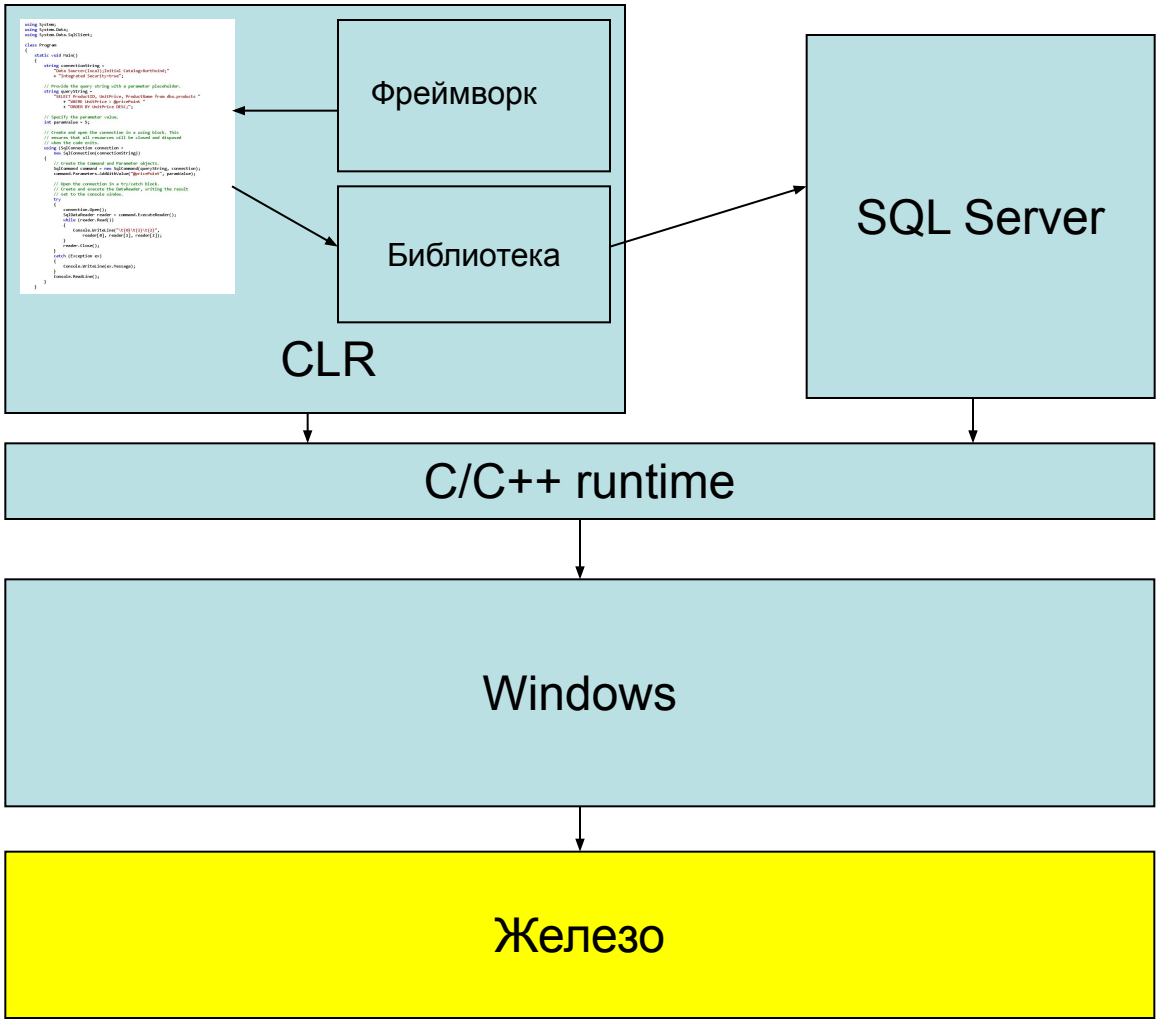
Оптимизирующий
Генератор байт-кода

JIT-компилятор

STL/ATL, препроцессор,
Оптимизирующий компилятор

VMWare патчит
гостевую ось на ходу

Железо руками давно уже
никто не проектирует,
Везде сплошные
CAD, VHDL и FPGA



И как с этим теперь жить?

- «Я лучше буду знать что-то одно, но буду знать это хорошо»
- «Пусть за меня думает компьютер, я буду решать только высокоуровневые вопросы»
- «Преждевременная оптимизация – корень всех зол»
- «Будет тормозить – будем думать»

Протекающие абстракции

- Leaky abstraction © Joel Spolsky, http://www.joelonsoftware.com/articles/LeakyAbstractions.html
- Что же может протечь?
  - Проблемы с безопасностью
  - Проблемы с производительностью
  - Ошибки
  - Документированные особенности реализации

Например

- Transmission Control Protocol
  - Абстракция надежного соединения через ненадежную сеть
- Реальная сеть теряет пакеты или доставляет их не по порядку
- TCP имитирует надежное соединение

Но

- TCP использует подтверждения, таймауты и повторную передачу
- TCP рвет соединение при трех последовательных таймаутах
- TCP гарантирует доставку, но не гарантирует время доставки

Другой пример

- Java/C# используют сборку мусора
- Полезная абстракция, которая устраняет висячие ссылки и утечки памяти
- Снижение стоимости разработки от 2 до 5 раз

Но

- Если вы не думаете, куда (а также, когда и почему) у вас деваются объекты, вы будете делать ошибки.
- В C++ и C# это будут разные ошибки.
- Кстати, в Java/C# утечки памяти очень даже бывают…

Exception security

C++ style

RAII (Resource allocation is initalization)

Деструкторы зовутся в конце блока

Завернуть выделяемый ресурс в объект с конструктором/деструктором, и исключения вам не страшны

Java/C# style

Финализаторы зовутся JVM знает когда

Внешние ресурсы надо освобождать явно

Висячие ссылки/утечки возвращаются к нам

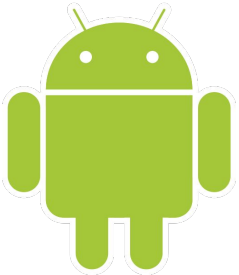
Try {}

Catch {}

Catch {}

Finally { window.close(); bitmap.recycle(); session.destroy(); …. }

Пример из жизни



- В Android есть две кучи
  - Managed heap (Dalvik)
  - Native heap (OpenGL/BMP)
- Объектам нативной кучи надо явно звать recycle.
- Куча приколов
  - Нативная куча кончилась, вы зовете gc, а толку нет
  - Утечки памяти в нативной куче
  - Висячие ссылки в управляемой куче
  - И прочие радости жизни
- И все это в условиях жестко ограниченной памяти.

Пример из жизни (самый противный вариант)

- Класс-обертка вокруг битмапа в нативной куче
- Битмап большой, обертка маленькая
- Насоздавали битмапов, нативная куча кончилась, а управляемая куча почти не занята.
- Сборка мусора не проходит (не видит, чего собирать-то)

Пример из жизни (продолжение)

Ну давайте везде звать recycle()?

http://masterden.livejournal.com/50326.html

[]

Поэтому при работе с битмапами постоянно ставишь recycle. А вот мы и подобрались к объяснению баги, описанной в начале. Даже в доках от гугла есть пример отдачи картинки в OpenGL, где после трансфера её туда битмап "ресайклится". Ага! У меня в коде было тоже самое! Но тот самый злосчастный экран отличался от остальных тем, что битмапы грузил не с SD-карты, а из ресурсов прилады! А они (ресурсы) грузятся один раз на всё приложение. И, будучи пересайкленной, битмапа из ресурсов оставалась таковой до рестарта приложения

[]

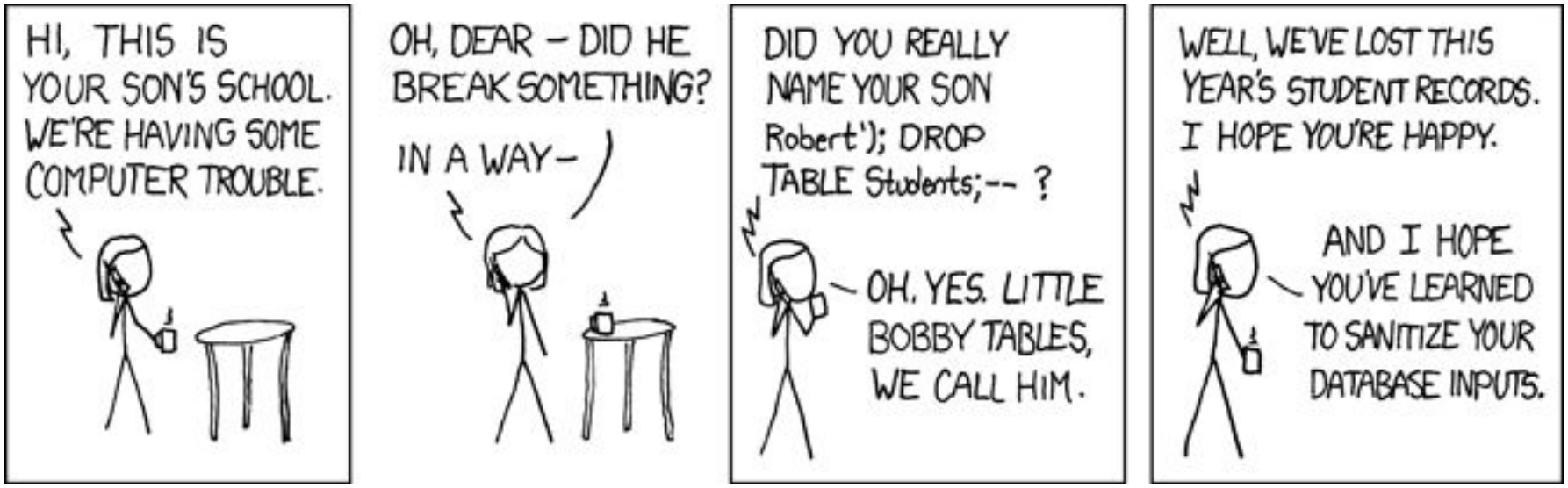
На самом деле все еще хуже

- Приколы вылезают не только при явном освобождении памяти
- Приколы могут вылезать если вы используете два разных сборщика мусора
- Например:
  - CLR (.Net) использует mark’n’sweep
  - COM (legacy VB) использует reference count

В чем разница?

- Mark’n’sweep
  - Сборка мусора дорогая операция
  - Зовется только от горя (когда память кончилась)
- Refcount
  - Сборка мусора происходит «сама», звать ее не надо
  - Никаких хуков на «что-то мы сожрали много памяти» нету
- Тот же сценарий: маленькая обертка в mark’n’sweep куче вокруг большого refcount объекта
  - Refcount куча разрослась до хрен-знает-скольколлиона байт, а mark’n’sweep про это ни сном, ни духом

Безопасность



Dynamic SQL, говорите?

```
using System.Data;
using System.Data.SqlClient;

using (SqlConnection connection = new SqlConnection(connectionString))
{
  DataSet userDataset = new DataSet();
  SqlDataAdapter myDataAdapter = new SqlDataAdapter(
        "SELECT au_lname, au_fname FROM Authors WHERE au_id = @au_id",
        connection);
  myCommand.SelectCommand.Parameters.Add("@au_id", SqlDbType.VarChar, 11);
  myCommand.SelectCommand.Parameters["@au_id"].Value = SSN.Text;
  myDataAdapter.Fill(userDataset);
}
```

Хех…

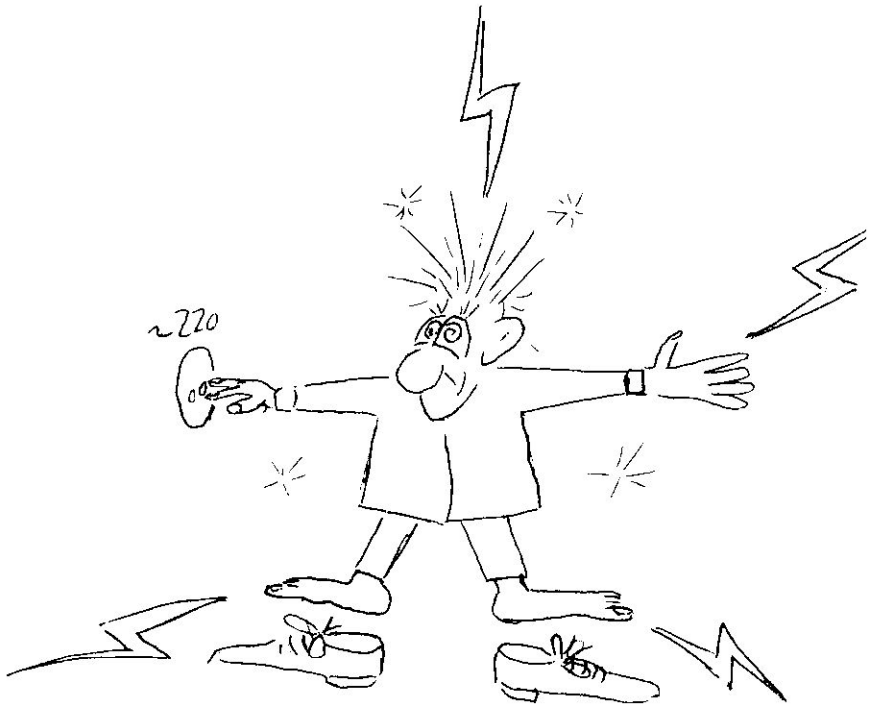
Кавычка попадает в базу → Legacy код на VB/ASP/Abscess берет ее из базы →

Хорошие практики

- Ну и что? – скажут многие
- Не надо знать низкоуровневые детали и не надо знать, что может протечь
- Надо знать хорошие практики, как бороться с протечками

Хорошие практики не панацея

- Хорошая практика: нельзя совать пальцы в розетку



Во что это может вылиться