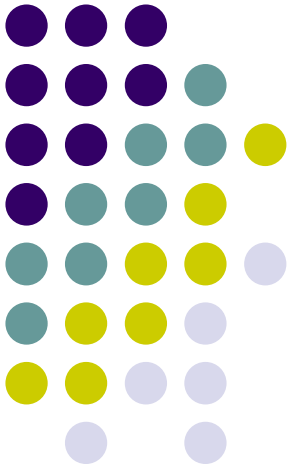
Object Oriented Programming with C#.NET

Lecturer: Prof. *Dr. E. Krustev* (**eck@fmi.uni-sofia.bg**)



Books:

[1] **P. J. Deitel, H. M. Deitel, "**C# 6 for Programmers**",** **6th ed., Prentice Hall 2017, ISBN-13: 978-0-13-459632-7 ISBN-10: 0-13-459632-3** (**основна**)

[2] **Daniel Solis**, **Cal Schrotenboer** "Illustrated C# 7", 5th ed. **APress 2018, ISBN-13 (pbk): 978-1-4842-3287-3**

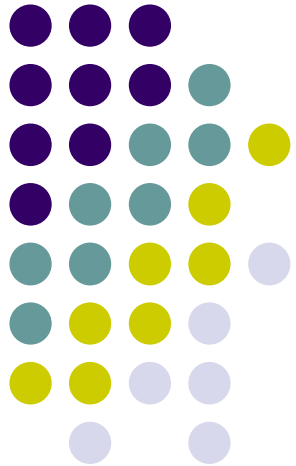
[3] **John Sharp,** "Microsoft Visual C# Step By Step", **Pearson education**, **2018 ISBN-13: 978-1-5093-0776-0, ISBN-10: 1-5093-0776-1**

Object Oriented Programming with C#.NET

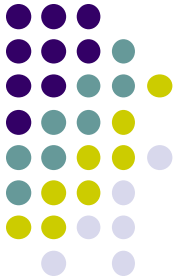
Software

Visual Studio .NET 2015- 2017

Blend for Visual Studio 2017

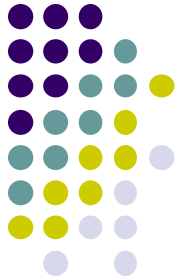


Main Topics



1. Създаване на съвременни обектно ориентирани приложения с използване на наследственост и полиморфизъм.
2. Изграждане **на интерактивен графичен интерфейс** като се **акцентира на създаване на приложения** с **WPF**.
3. Създаване на потребителски компоненти и библиотеки
4. Структури от данни с приложения на **LINQ, Parallel LINQ** и **Task Parallel Library.**
5. Многонишково програмиране.

Main Topics



6. Обработка на файлове и потоци от данни със сериализация на обекти при сървър-клиент приложения
7. Уеб услуги с приложение на WCF за реализация на SOAP Web Services с .NET

Акцентира се върху техники за добър стил и ефективност на програмния код, съобразени с **.NET Framework**

Evaluation

Final grade components:

Written examination during the term

- **A written midterm exam (40%)**
- **A Course project (20%)**

Written examination after the term end

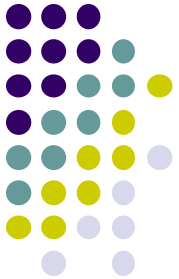
- **A final written exam (40%)**

Evaluation scale

American system:

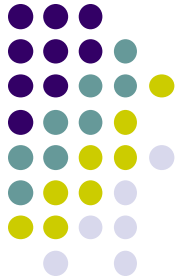
2. **от 0 до 54 точки**
3. **от 55 до 64 точки**
4. **от 65 до 74 точки**
5. **от 75 до 84 точки**
6. **от 85 до 100 точки**

Introduction to C#.NET



- ***Areas of application development***
- ***Overview of the C# 7 Language***
- ***Developing for the .NET Platform***

Introduction

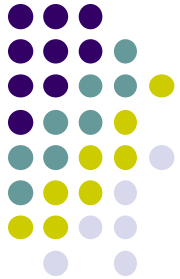


Objective:

Make you familiar with the basics of the **.NET Framework** and the process of **developing C# applications**, as well as, outline the **new features** in the **IDE** and the **C# language**

- C# is a **modern object oriented language** for developing a wide range of **secure and robust applications** that run on the .NET Framework
- Use **C#** to create ***traditional Windows client applications***, ***XML Web services***, **distributed components**, ***client-server applications***, ***database applications*** and many **more**

Areas of application development



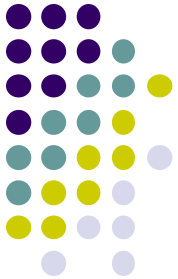
C# is an answer to a new problem:

developers need **a language that works well in a distributed programming environment**.

C# is a vast improvement over previous languages and it has many **new features** to offer in comparison to these languages

C# provides **capabilities for *distributed computing in web applications* and syntax similar to Java** enriched with a number of new important features.

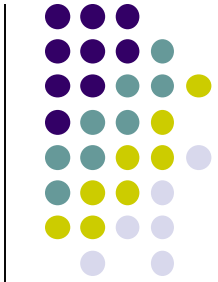
Advantages of C#.NET



C# is a vast improvement over previous languages (Java, VC++, VB)

- **low-level functionality of C++**
- **programming environment of Visual Basic**
- **capabilities for distributed computing in web applications and syntax similar to Java**

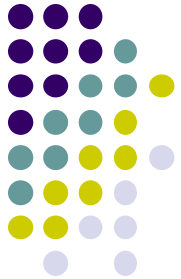
Advantages of C#.NET - **Summary**



- **C# is not a one language solution to all application problems**

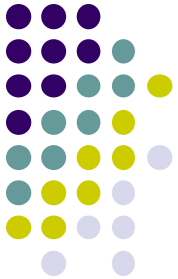
- **It has its limitations and its advantages are best established in its target area- distributed computing**

Developing for the .NET Platform- Summary



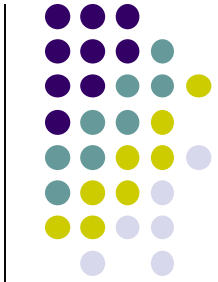
- **Software development independence from specific language or platform**
- **Portability of .NET programs across multiple platforms**
- **The concept of software reuse is extended to the Internet.**
- **Designed for use in a distributed application environment**

The C# Language - **Overview**



- **C# language is highly expressive in implementing all the features of a modern object-oriented programming language**
- **Provides support for encapsulation, inheritance, and polymorphism**

A FIRST C#.NET PROGRAM

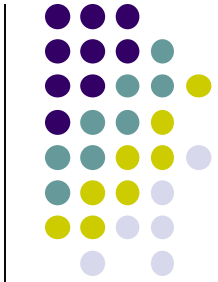


```
using System;

namespace HelloCSharp
{    // file : Hello.cs
     // compile : csc Hello.cs

     class Hello {
       public static void Main() {
           Console.WriteLine("Happy Programming!");
           Console.WriteLine("Press any key to quit ");
           Console.ReadLine();
       }
     }
}
```

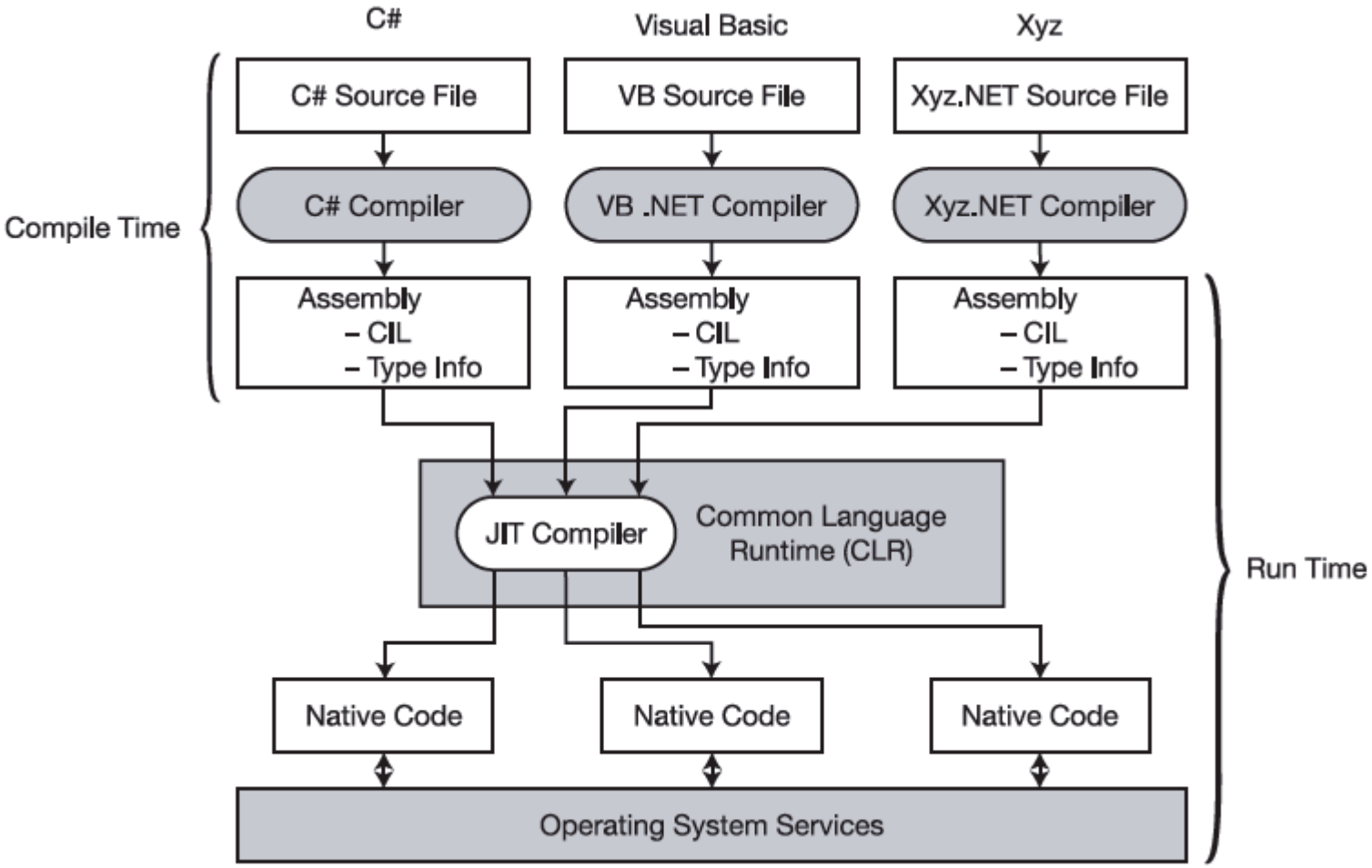
A FIRST C#.NET PROGRAM- Summary



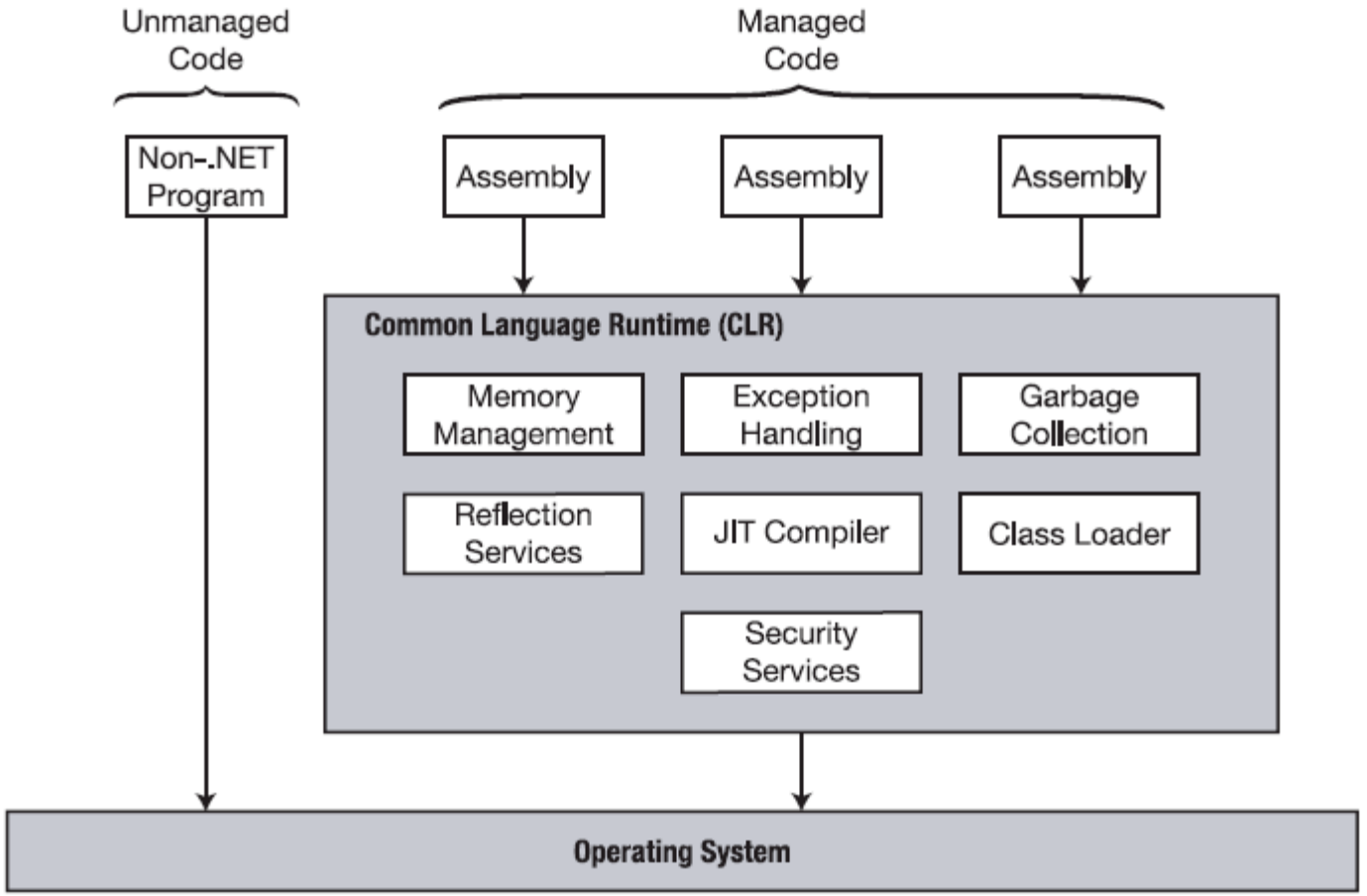
Remarks:

- **Every C# program must contain at least one class**
- **A Console application must have a `Main` method, which is the program's entry point where execution begins**
- **.NET Framework classes are grouped by function and logically arranged in namespaces**
- **Visual Studio.NET provides a user friendly IDE for rapid application development**

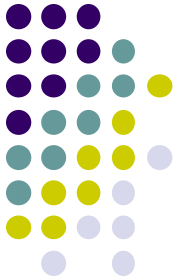
C#.NET software development lifecycle



C#.NET software development lifecycle

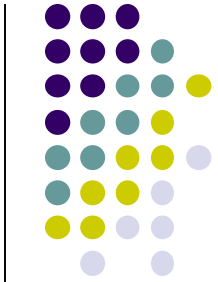


DEVELOPING FOR THE .NET PLATFORM



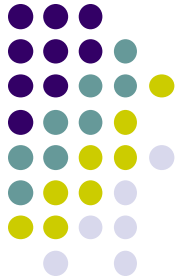
- **.NET is independent of any programming language**
- **Available are C#, Visual Basic .NET, JScript .NET, COBOL, Perl, Python, Eiffel, APL**
- **.NET understands only one language, Microsoft *Intermediate Language* (IL or MSIL).**

A language- independent platform



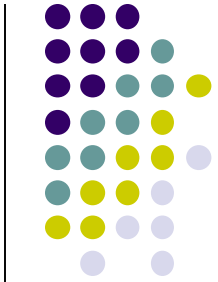
- **The compilation process produces a Windows executable file in *portable executable* (PE) format, which is later on used by the *Common Language Runtime (*CLR) to run it**
- **CLR includes a language-neutral type system with support for classes, inheritance, polymorphism, dynamic binding, memory management, garbage collection, exception handling.**
- The **CLR** provides a common bridge to **facilitate language interoperation** and component integration.

A language- independent platform- MSIL



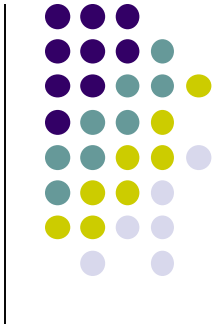
- **MSIL provides portability to the .NET Framework and is also the key to the framework's language interoperability.**
- **MSIL can be described as *an assembly language for a stack-based, virtual, .NET "CPU."***
- **Before the application gets executed, the CLR performs another compilation known as Just- In-Time (JIT) compilation to native machine code.**
- **Implications. First, the CLR neither knows, nor cares, what language was used to create the application or component. It just sees MSIL. Second, in theory, replacing the JIT compiler is all that's necessary to target a new platform.**

A language- independent platform- MSIL



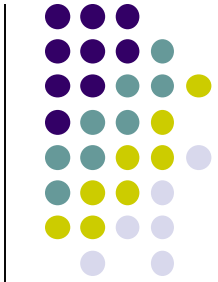
- **MSIL is always fully compiled before it is executed.**
- **Each method gets compiled *once* as it gets called within a program.**
- **.NET does not use a virtual machine (as Java) to execute the program.**

Sample MSIL



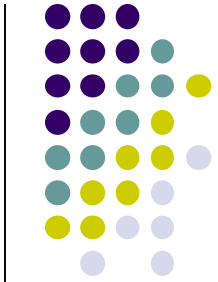
```
.method public hidebysig static void  Main() cil managed
{
  .entrypoint
  // Code size       28 (0x1c)
  .maxstack  1
  IL_0000:  ldstr      "Happy Programming with C#.NET!"
  IL_0005:  call       void [mscorlib]System.Console::WriteLine(string)
  IL_000a:  pop
  IL_000b:  ldstr      "Press any key to close the application.. "
  IL_0010:  call       void [mscorlib]System.Console::WriteLine(string)
  IL_0015:  call       string [mscorlib]System.Console::ReadLine()
  IL_001a:  pop
  IL_001b:  ret
} // end of method Hello::Main
```

A language- independent platform- MSIL



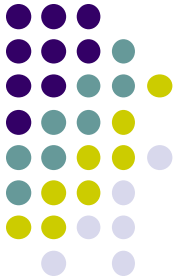
- **MSIL can be read with MSIL Disassembler (*ildasm.exe*) .**
- **It is comprised of an instruction set and an array of features that are designed to support the essential operations and characteristics of many modern, object-oriented languages .**
- **MSIL code is actually quite easy to read and understand**

A language- independent platform- MSIL Summary



- **MSIL** was **not designed with a particular programming language** in mind.
- **MSIL statements** manipulate **common types shared by all .NET languages**. This is known as the ***Common Type System***, or **CTS**..
- To facilitate **cross-language interoperability, .NET** includes a ***Common Language Specification***, or **CLS**, that represents a common standard to which **.NET** types should adhere. This standard lays down **rules** relating to **allowed primitive types, array bounds, reference types, members, exceptions, attributes, events, delegates,**

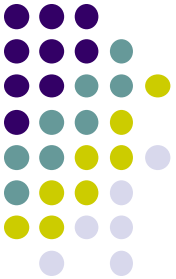
Modified Hungarian Notation



- It is very important to **keep the coding style consistent.**
- **short prefix mnemonics** that allowed programmers to easily identify the type of information a variable might contain .
- both types of code **interoperate**

Modified Hungarian Notation

Some commonly used prefixes in this course:

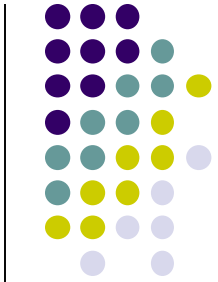


Control	Prefix
Button	*btn*
ComboBox	*cbo*
CheckBox	*chk*
Label	*lbl*
ListBox	*lst*
MainMenu	*mnu*
RadioButton	*rdb*
PictureBox	*pic*
TextBox	*txt*

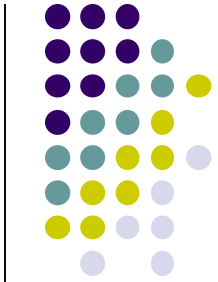
Modified Hungarian Notation

As a **general rule**, notice that in **C#.NET**:

- **class and interface** names start by a **Capital** letter
- **references** to classes and interfaces, as well, as **variables of primitive data types** such as ***int, bool double*** etc start by a **lowercase letter**
- the **names of methods** start by a **Capital** letter
- the **names of controls have to follow the Modified Hungarian notation** explained above (*they have to be descriptive by means of introducing appropriate prefixes*)



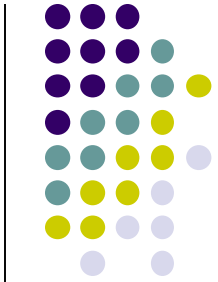
Writing good code



Good programming qualities:

- **Simplicity**
- **Readability**
- **Modularity**
- **Layering**
- **Design**
- **Efficiency**
- **Elegance**
- **Clarity**

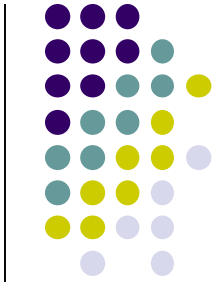
Writing good code



Simplicity

Means you ***don't do in ten lines what you can do in five.*** It means you make **extra effort to be concise**, but not to the point of obfuscation. It means you **abhor open coding and functions that span pages**. Simplicity- of organization, implementation, design- makes your **code more reliable and bug free**. There's less to go wrong

Writing good code

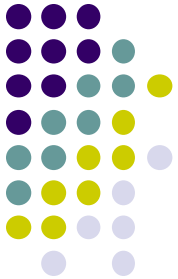


Readability

Means what it says: ***that others can read your code***.
Readability means you bother to **write comments, to follow conventions**, and pause to **name your variables wisely**.
Like choosing "*taxrate*" instead of "*tr*".

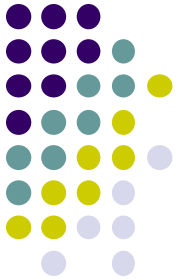
Writing good code

Modularity



Means ***your program is built like the universe***. The world is made of molecules, which are made of atoms, electrons, nucleons, quarks, and (if you believe in them) strings. Likewise, **good programs erect large systems from smaller ones**, which are built from even smaller building blocks. You can write a text editor with three primitives: move, insert, and delete. And **just as atoms combine in novel ways, software components should be reusable.**

Writing good code

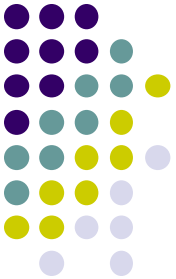


Layering

Means that **internally, your program resembles a layer cake**. The app sits on the framework sits, the OS sits on the hardware. Even within your app, you need layers, like file-document-view-frame. **Higher layers call ones below, which raise events back up**. (Calls go down; events go up.) **Lower layers should never know what higher ones are up to.** The essence of an **event/callback** is to provide **blind upward notification**. .

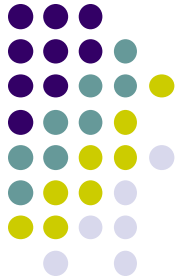
Writing good code

Design



Means you **take time to plan your program before you build it**. Thoughts are cheaper than debugging. **A good rule of thumb is to spend half your time on design**. You need a functional spec (what the programs does) and an internal blueprint. APIs should be codified in writing.. .

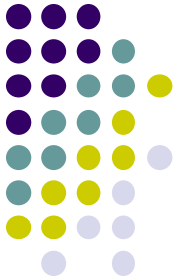
Writing good code



Efficiency

Means **your program is *fast* and *economical*. It doesn't hog files, data connections, or anything else**. It **does what it should, but no more**. It **loads and departs without fuss**. At the function level, you can always optimize later, during testing. But at high levels, you must **plan for performance**. If the design requires a million trips to the server, expect a big problem. .

Writing good code

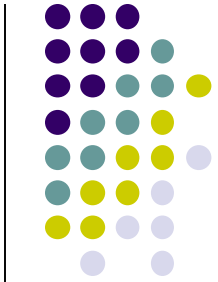


Ellegance

Elegance is like beauty: hard to describe but easy to recognize. Elegance combines ***simplicity***, ***efficiency***, **and *brilliance*, and produces a feeling of pride**. Elegance is when you replace a procedure with a table, or realize that you can use recursion- which is almost always elegant:

```
int fact(int n) {
    return n==0 ? 1 : n * fact(n-1);
 }
```

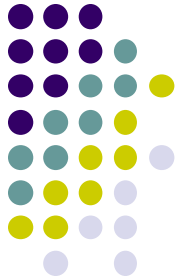
Writing good code



Clarity

Clarity is the platinum quality all the others serve. The fundamental challenge of programming is **managing complexity**. ***Simplicity, readability, modularity, layering, design, efficiency, and elegance*** **are all time-honored ways to achieve clarity**, which is the antidote to complexity. **You must understand- really understand- what you're doing at every level**. Otherwise you're lost. **Bad programs are less often a failure of coding skill than of having a clear goal.**

Happy Programming



OOP

C#.NET 2017