.NET Framework 2.0
Visual Studio 2005

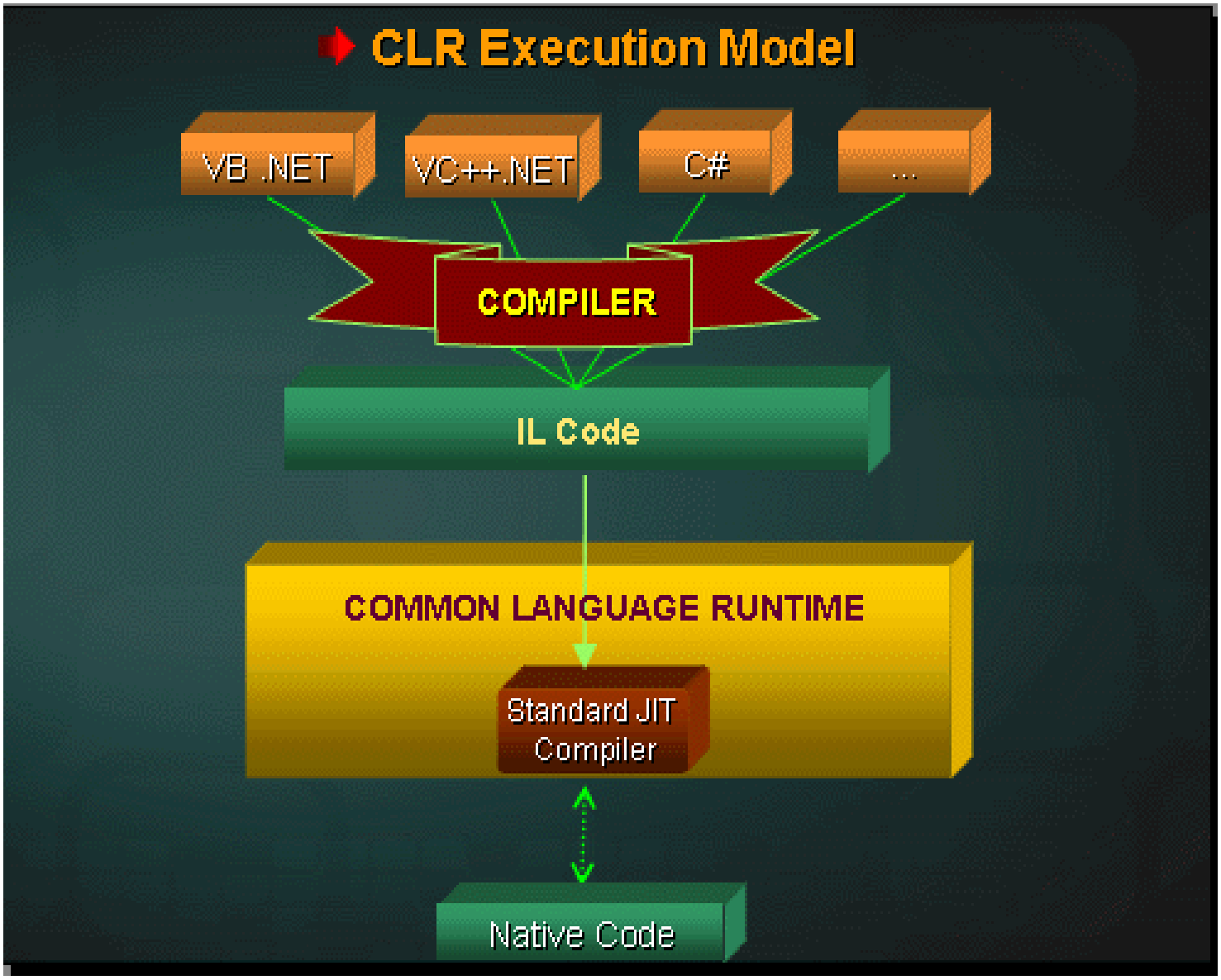
Wojciech Rygielski

.NET - Java

- .NET Framework
- Visual Studio
- MSIL
- C#, VB

- JRE
- Eclipse
- Bytecode
- Java

```
open System
open System.Windows.Forms

let form = new Form()
do form.Width  <- 400
do form.Height <- 300
do form.Text <- "Hello World Form"

(* Menu bar, menus *)
let mMain = form.Menu <- new MainMenu()
let mFile = form.Menu.MenuItems.Add("&File")
let miQuit  = new MenuItem("&Quit")
let _ = mFile.MenuItems.Add(miQuit)

(* RichTextView *)
let textB = new RichTextBox()
do textB.Dock <- DockStyle.Fill
do textB.Text <- "Hello World\n\nCongratulations!"
do form.Controls.Add(textB)

(* callbacks *)
let opExitForm sender args = form.Close ()
do miQuit .add_Click (new EventHandler(opExitForm))

(* run! *)
do Application.Run(form)
```


CLR Execution Model
VB .NET
VC++.NET
C#
...
COMPILER
IL Code
COMMON LANGUAGE RUNTIME
Standard JIT Compiler
Native Code

Języki

- Ada
- APL
- AsmL
- BrainFuck
- C Standard
- C#
- C++
- Caml
- Cobol
- Delphi
- Eiffel
- Forth

Języki

- Fortran
- Haskell
- Java
- JavaScript
- LOGO
- Lua
- Mercury
- Mondrian
- Oberon
- Pascal
- Perl
- PHP

Języki

- PL/1
- Prolog
- Python
- RPG
- Ruby
- Scala
- Scheme
- Small Talk
- SML (Standard Meta Language)
- Visual Basic


Zmiany w .NET

Partial types

```
public partial class EmployeeInfo
{
        // metoda 1
}

public partial class EmployeeInfo
{
        // metoda 2
}
```

Anonimowe metody

```
// Dotychczas:

public partial class Form1 : Form
{
	public Form1()
	{
		InitializeComponent();
		Button b = new Button();
		b.Click += new EventHandler(b_Click);
		this.Controls.Add(b);
	}

	void b_Click(object sender, EventArgs e)
	{
		this.Close();
	}
}
```

Anonimowe metody

```
// Teraz:

public partial class Form1 : Form
{
	public Form1()
	{
		InitializeComponent();
		Button b = new Button();
		b.Click += delegate(object s, EventArgs e)
		{
			this.Close();
		};
		this.Controls.Add(b);
	}
}
```

Co to delegaty?

```
class Book { /*... */ }
delegate void BookProcessingProc(Book book);

class Library
{
        public event BookProcessingProc BookAdded;
        /* ... */
}

static void Main(string[] args)
{
        Library lib = new Library();
        lib.BookAdded += new BookProcessingProc(lib_BookAdded);
}

static void lib_BookAdded(Book book)
{
        /* ... */
}
```

```
class Book { /*... */ }
delegate void BookProcessingProc(Book book);

class Library
{
        public event BookProcessingProc BookAdded;
        public void OnBookAdded(Book b)
        {
                if (this.BookAdded != null)
                        this.BookAdded(b);
        }
}

static void Main(string[] args)
{
        Library lib = new Library();
        lib.OnBookAdded(new Book());
        lib.BookAdded += new BookProcessingProc(lib_BookAdded);
        lib.BookAdded += delegate(Book b) { Console.WriteLine("X"); };
        lib.OnBookAdded(new Book());
        Console.ReadLine();
}

static void lib_BookAdded(Book book)
{
        Console.WriteLine("obsługa eventa");
}
```

Generics

```
static ArrayList kolekcja = new ArrayList();
static void Main(string[] args)
{
        kolekcja.Add(new System.IO.FileInfo("blablabla"));
        kolekcja.Add(666);
        Console.WriteLine(((System.IO.FileInfo)kolekcja[0]).Length);
}
```

```
static List<System.IO.FileInfo> kolekcja =
         new List<System.IO.FileInfo>();
static void Main(string[] args)
{
         kolekcja.Add(new System.IO.FileInfo("blablabla"));
         // kolekcja.Add(666);   <- błąd kompilacji
         Console.WriteLine(kolekcja[0].Length);
}
```

Generics

```
public class Stack<T>
{
        T[] items;
        int count;
        public void Push(T item)    {...}
        public T Pop() {...}
}

Stack<int> stack = new Stack<int>();
stack.Push(3);
int i = stack.Pop();
```

Generics

```
interface IComparable <T>

struct HashBucket <K,D>

static void Reverse <T> (T[] arr)

delegate void Action <T> (T arg)
```

Templates vs Generics

- Templates are instantiated at compile-time with the source code.
- Templates are type safe.
- Templates allow user-defined specialization.
- Templates allow non-type parameters.
- Templates use "lazy structural constraints".
- Templates support mix-ins.

- Generics are instantiated at run-time by the CLR.
- Generics are also type safe.
- Generics are cross-language.
- Generics do not allow user-defined specialization.
- Generics do not allow non-type parameters.
- Generics use subtype constraints.

Templates vs Generics

```
// static bool CzyMniejsza<T>(T a, T b)
// {
//     return a < b;
// }

static bool CzyMniejsza<T>(T a, T b)
      where T : IComparable
{
      return a.CompareTo(b) < 0;
}

static void Main(string[] args)
{
      bool wynik = CzyMniejsza<int>(3, 6);
}
```

Enumeratory (yield)

```
static IEnumerator<int> Fib()
{
	yield return 1;
	int a = 0;
	int b = 1;
	int t;
	while (true)
	{
		t = b;
		b = b+a;
		a = t;
		yield return b;
	}
}
```

Enumeratory (yield)

```
static IEnumerator<int> Ciąg() {
        for (int i=1; i<7; i++)
        {
                yield return 1;
                yield return 2;
                if (i==3)
                        yield break;
        }
}

static void Main(string[] args)
{
        IEnumerator<int> e = Ciąg();
        while (e.MoveNext())
                Console.Write(e.Current); // „121212”
        Console.ReadLine();
}
```

Nullable value types

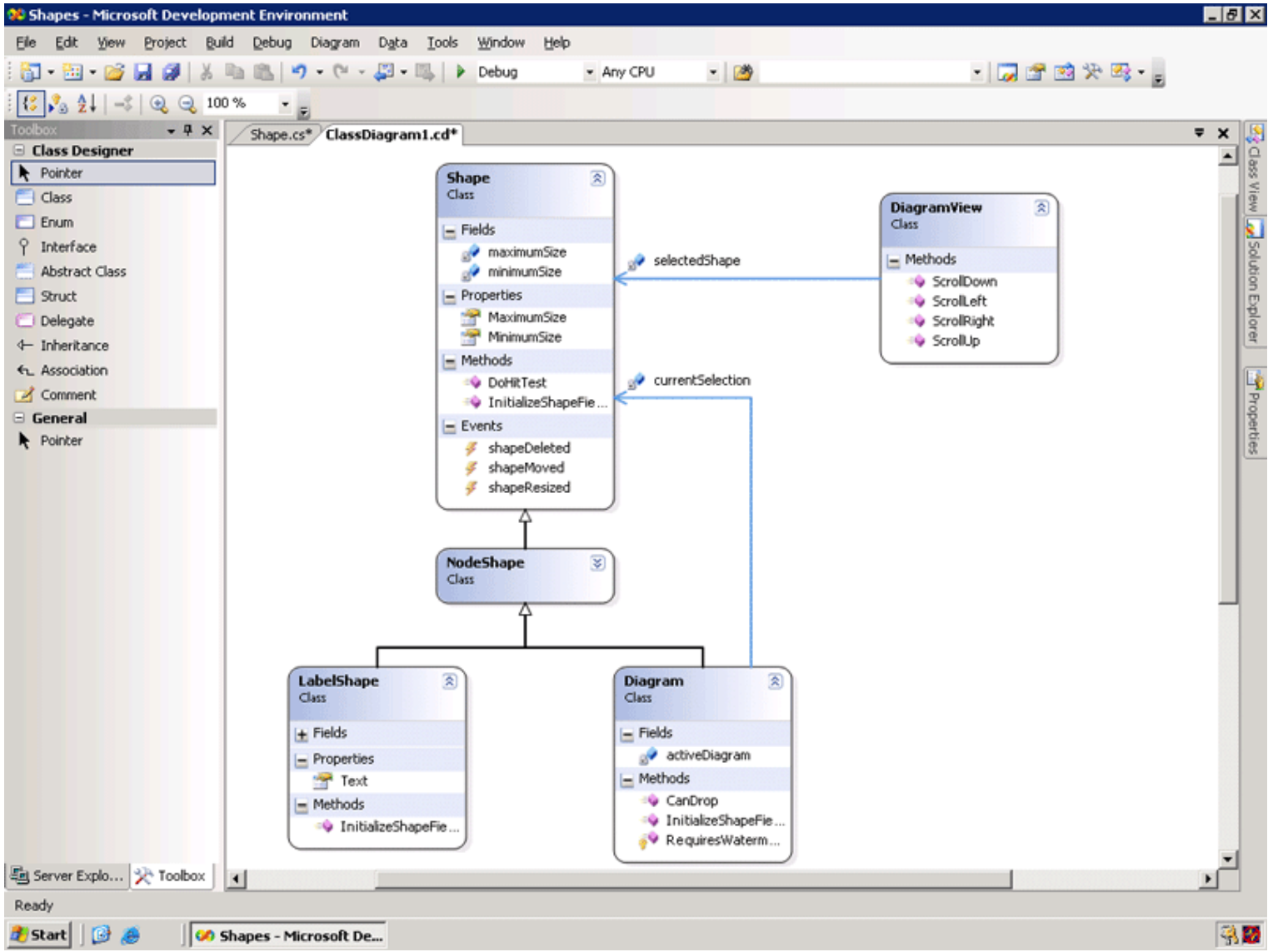
```
static int? DajLiczbe()
{
        return null;
}

static void Main(string[] args)
{
        int? n = DajLiczbe();
        // if (n!=null)
        //     Console.WriteLine(n);
        if (n.HasValue)
                Console.WriteLine(n.Value);
}
```

Nullable value types

```
static int? DajLiczbe()
{
	return 3;
}

static void WypiszLiczbe(int n)
{
	Console.WriteLine(n);
}

static void Main(string[] args)
{
	WypiszLiczbe(DajLiczbe()); // błąd kompilacji
}
```

Nowości w VS

- MenuStrip, ContextMenuStrip, ToolStrip
- TableLayout, FlowLayout
- ClickOnce
- Visualizers
- Class diagram
- Debugging Code at Design Time
- TracePoints

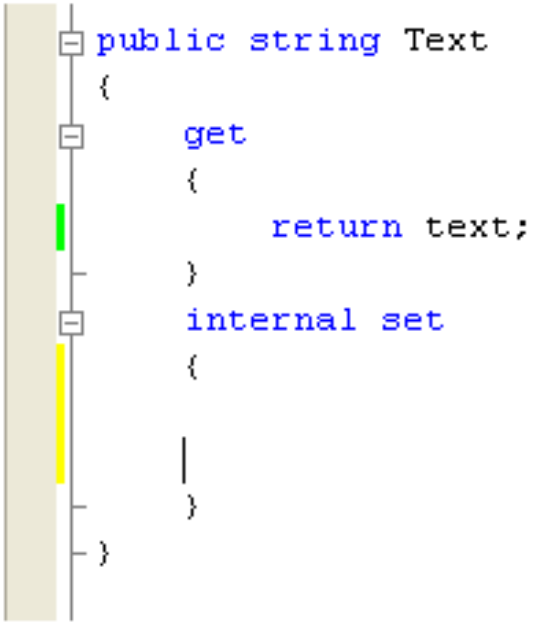

Shapes - Microsoft Development Environment
File Edit View Project Build Debug Diagram Data Tools Window Help
Debug
Any CPU
100 %
Toolbox
Class Designer
Pointer
Class
Enum
Interface
Abstract Class
Struct
Delegate
Inheritance
Association
Comment
General
Pointer
Shape.cs*
ClassDiagram1.cd*
Shape
Class
Fields
maximumSize
minimumSize
Properties
MaximumSize
MinimumSize
Methods
DoHitTest
InitializeShapeFie...
Events
shapeDeleted
shapeMoved
shapeResized
selectedShape
currentSelection
DiagramView
Class
Methods
ScrollDown
ScrollLeft
ScrollRight
ScrollUp
NodeShape
Class
LabelShape
Class
Fields
Properties
Text
Methods
InitializeShapeFie...
Diagram
Class
Fields
activeDiagram
Methods
CanDrop
InitializeShapeFie...
RequiresWaterm...
Class View
Solution Explorer
Properties
Server Explo...
Toolbox
Ready
Start
Shapes - Microsoft De...

Snippets

```
<?xml version="1.0" encoding="utf-8" ?>
<CodeSnippet Format="1.0.0">
   <Header>
      <Title>class</Title>
      <Shortcut>class</Shortcut>
      <Description>Expansion snippet for class</Description>
      <SnippetTypes>
         <SnippetType>Expansion</SnippetType>
         <SnippetType>SurroundsWith</SnippetType>
      </SnippetTypes>
   </Header>
   <Snippet>
      <Declarations>
         <Literal default="true">
            <ID>name</ID>
            <ToolTip>Class name</ToolTip>
            <Default>MyClass</Default>
         </Literal>
      </Declarations>
      <Code Language="csharp" Format="CData"><![CDATA[class $name$
   {
      $selected$$end$
   }]]>
   </Code>
   </Snippet>
</CodeSnippet>
```

Refactoring

- Extract Method: This is to split a method into many fine grained methods which are reusable.
- Rename: This is used to rename an identifier i.e. field, variable, method etc.
- Encapsulate Fields: Creating a property to encapsulate a field.
- Extract Interfaces: Creating an interface which is implemented by current class.
- Promote Local Variable to Parameters: Moving a local variable to parameter level.
- Remove Parameters: Removing parameters from methods, indexers, constructors, delegates.
- Reorder Parameters: Changing the order of parameters.
- Generate Method Stub: Automatic code generation based on the consumption of a method.
- Add Using Unbound Types: Automatic inclusion of namespace for an unbound type.


public string Text
{
get
{
return text;
}
internal set
{
}
}


C# 3.0

Language integrated query (LINQ)

```
public void Linq1() {
    int[] numbers = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };

    var lowNums =
        from n in numbers
        where n < 5
        select n;

    Console.WriteLine("Numbers < 5:");
    foreach (var x in lowNums) {
        Console.WriteLine(x);
    }
}
```

- Object initialization:

```
Customer c = new Customer { Name="James" };
```

- Lambda expressions:

```
listOfFoo.Where(x => x.size>10);
Func<int,int> f =  x => x + 1;
```

Źródła

- http://msdn.microsoft.com/
- http://research.microsoft.com/
- http://www.arunmicrosystems.netfirms.com/clr.html