EKON 14

Top Ten Metrics

10 print “hello EKON”;
20 Goto 10;

“Metrics measure the design of code after it has been written”

Kleiner
Kommunikation.....

Agenda EKON

- What are Metrics ?
- How to recognize Bad Code ?
- Top Ten (The Law of Demeter)
- Improve your Code (Metric Based Refactoring)
- Optimisation and other Tools

What are Metrics?

Metrics are for

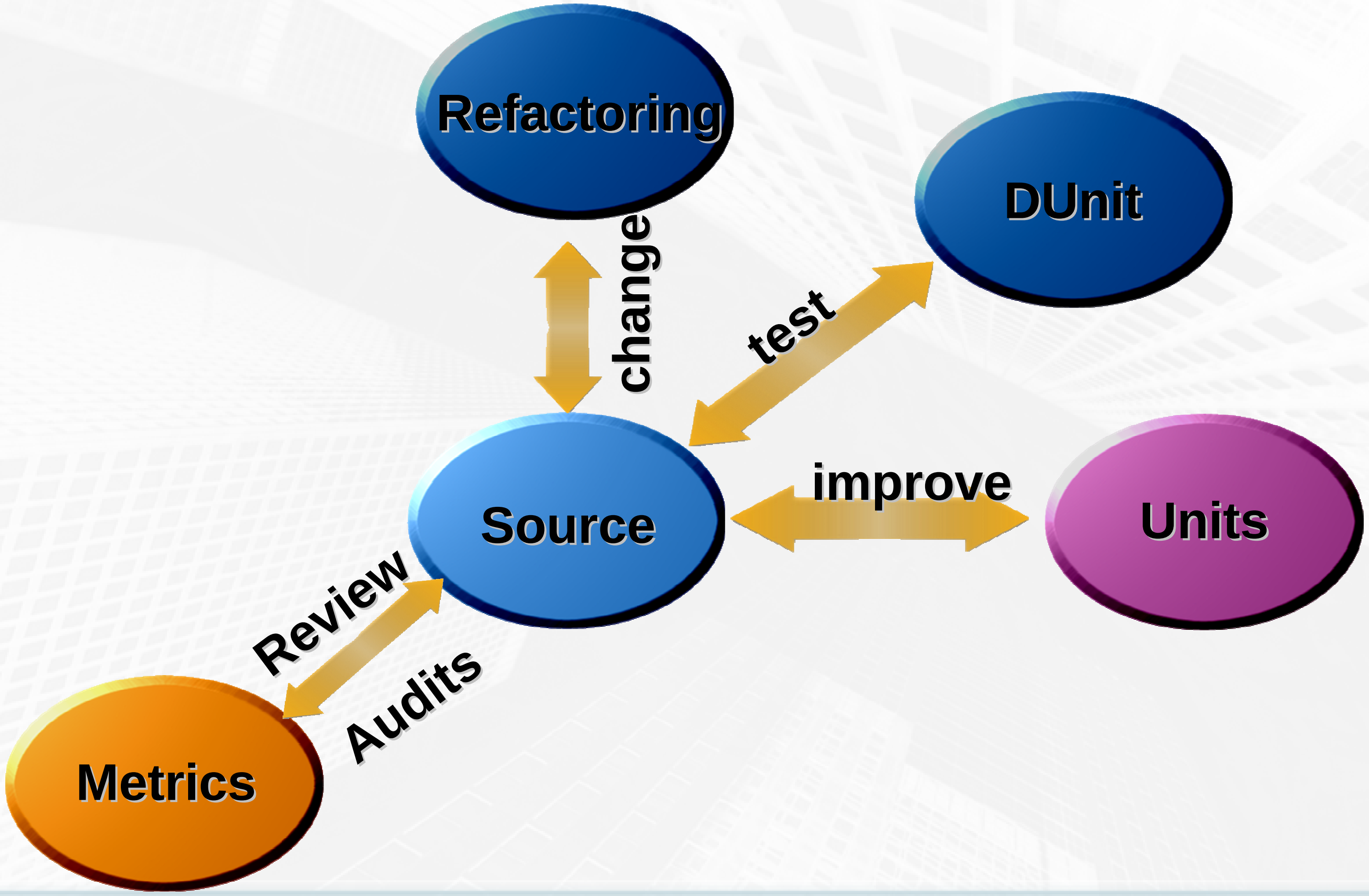
- Evaluate Object Complexity
- Quantify your code
- Highlight Redesign Needs
- Change Impact Analysis

Metrics deal with

Bad Structure

- General Code Size (in module)
- Cohesion (in classes and inheritance)
- Complexity
- Coupling (between classes or units)
  - Cyclic Dependency, Declare+Definition, ACD-Metric
- Interfaces or Packages (design & runtime)
- Static, Public, Private (inheritance or delegate)

Metric Context



Some Kind of wonderful ?

- statusbar1.simpletext
  - simplepanel:= true!
- TLinarBitmap = TLinearBitmap; //Spelling bug
- aus Win32.VCL.Dialogs.pas
  - WndProcPtrAtom: TAtom = 0;
- aus indy: self.sender!
  - procedure TIdMessageSender_W(Self: TIdMessage; const T: TIdEmailAddressItem);
  - begin Self.Sender := T; end;

When and why Metrics ?

After a Code Review

By changes of a release

Redesign with UML (Patterns or Profiles)

Law of Demeter not passed

Bad Testability (FAT or SAT)

- Work on little steps at a time
- Modify not only structure but also code format

What‘s Bad Code

Bad Coordination

- Inconsistence with Threads
- Access on objects with Null Pointer
- Bad Open/Close of Input/Output Streams or I/O Connections
- Check return values or idempotence
- Check break /exit in loops
- Modification of static or const code
- Access of constructor on non initialized vars

Metric/Review Checklist

1. Standards - are the Pascal software standards for name conventions being followed?
2. Are all program headers completed?
3. Are changes commented appropriately?
4. Are release notes Clear? Complete?
5. Installation Issues, Licenses, Certs. Are there any?
6. Version Control, Are output products clear?
7. Test Instructions - Are they any? Complete?
8. "Die andere Seite, sehr dunkel sie ist" - "Yoda, halt's Maul und iß Deinen Toast!"

Top Ten Metrics

1. VOD Violation of Law of Demeter
2. Halstead NOpmd (Operands/Operators)
3. DAC (Data Abstraction Coupling)(Too many responsibilities or references in the field)
4. CC (Complexity Report), McCabe cyclomatic complexity, Decision Points)
5. CBO (Coupling between Objects)→ Modularity

Top Ten II

6. PUR (Package Usage Ratio) access information in a package from outside
7. DD Dependency Dispersion (SS, Shotgun Surgery (Little changes distributed over too many objects or procedures → patterns missed))
8. CR Comment Relation
9. MDC (Module Design Complexity (Class with too many delegating methods)
10. NORM →(remote methods called (Missing polymorphism))

Law of Demeter

You should avoid:

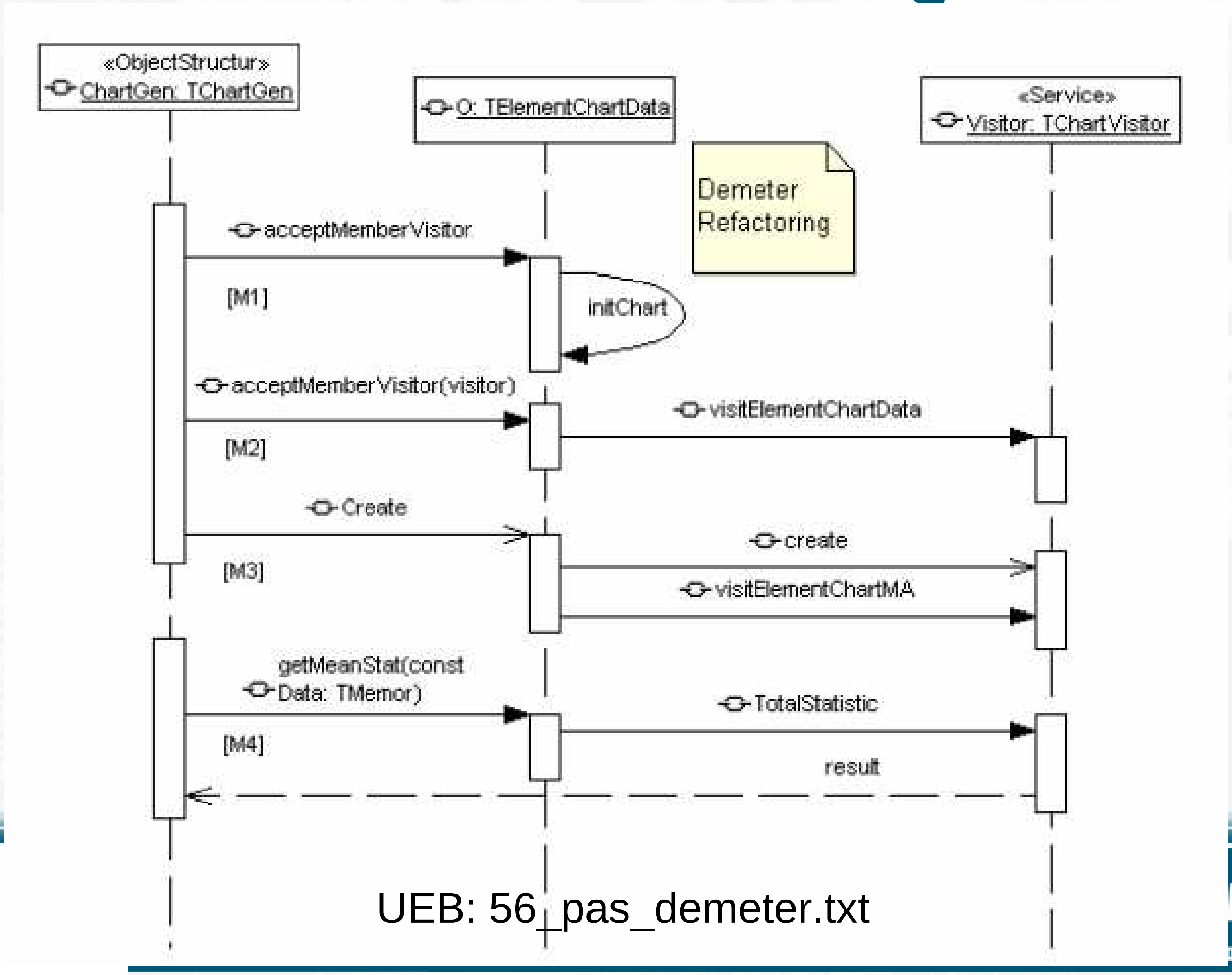
• Large classes with strange members

•***Das Gesetz von Demeter (don‘t talk to strangers) besagt, dass ein Objekt O als Reaktion auf eine Nachricht m, weitere Nachrichten nur an die folgenden Objekte senden sollte: (all objects to which m sends a message must be instances of classes)***

Demeter konkret

1. [M1] an Objekt O selbst
   Bsp.: self.initChart(vdata);
2. [M2] an Objekte, die als Parameter in der Nachricht m vorkommen
   Bsp.: O.acceptmemberVisitor(visitor)
   visitor.visitElementChartData;
3. [M3] an Objekte, die O als Reaktion auf m erstellt
   Bsp.: visitor:= TChartVisitor.create(cData, madata);
4. [M4] an Objekte, auf die O direkt mit einem Member zugreifen kann
   Bsp.: O.Ctnr:= visitor.TotalStatistic

Demeter Test as SEQ



UEB: 56_pas_demeter.txt

EKON

DAC or Modules of Classes

Large classes with to many references
- More than seven or eight variables
- More than fifty methods
- You probably need to break up the class in Components (Strategy, Composite, Decorator)

```
TWebModule1 = class(TWebModule)
    HTTPSoapDispatcher1: THTTPSoapDispatcher;
    HTTPSoapPascalInvoker1: THTTPSoapPascalInvoker;
    WSDLHTMLPublish1: TWSDLHTMLPublish;
    DataSetTableProducer1: TDataSetTableProducer;
```

CC

- Check Complexity

```
function IsInteger(TestThis: String): Boolean;
begin
  try
    StrToInt(TestThis);
  except
    on EConvertError do
      result:= False;
  else
    result:= True;
  end;
end;
```

CBO I

CBO measures the number of classes to which a class is coupled. According to remarks and comments on CBO and coupling, we include coupling through inheritance.

Two classes are considered coupled, if methods declared in one class call methods or access attributes defined in the other class.

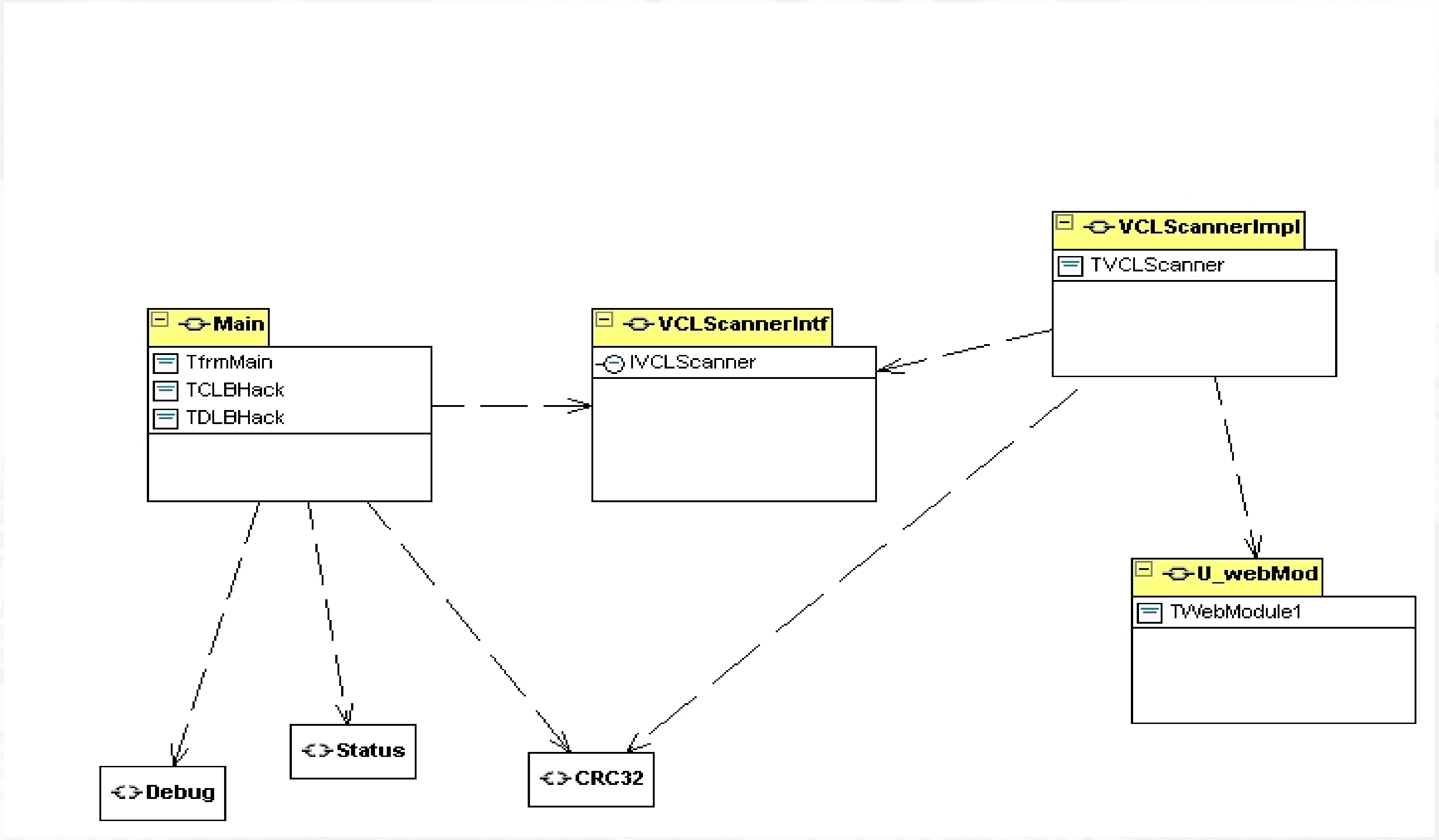
CBO II

Are Business Objects available for good CBO (Extensions)?

In a simple business object (without fields in the class), you do have at least 4 tasks to fulfil:

1. The Business-Class inherits from a Data-Provider
2. The query is part of the class
3. A calculation or business-rule has to be done
4. The object is independent from the GUI, GUI calls the object *“Business objects are sometimes referred to as conceptual objects, because they provide services which meet business requirements, regardless of technology”.*

PUR Package Usage Ratio



Welche Metric? → DD

- Dependency Dispersion (Code too much Distributed):

```
…..
for i:= 0 to SourceTable.FieldCount - 1 do
    DestTable.Fields[i].Assign(SourceTable.Fields[i]);
  DestTable.Post;
…..
```

DD – use small procedures

```
Procedure CopyRecord(const SourceTable, DestTable:
                                                        TTable);
var i: Word;
   begin
     DestTable.Append;
     For i:= 0 to SourceTable.FieldCount - 1 do
       DestTable.Fields[i].Assign(SourceTable.Fields[i]);
     DestTable.Post;
   end;
```

Finally you can measure:

Bad Naming (no naming convention)

Duplicated Code (side effects)

Long Methods (to much code)

Temporary Fields (confusion)

Long Parameter List (Object is missing)

Data Classes (no methods)

- Large Class with too many delegating methods

In a Kiviat Chart you get a Best Practices Circle!

Why is Refactoring important?

- Only defense against software decay.
- Often needed to fix reusability bugs.
- Lets you add patterns or templates after you have written a program;
- Lets you transform program into framework.
- Estimation of the value (capital) of code!
- Necessary for beautiful software.

Refactoring Process

The act of serialize the process:

- Build unit test
- Refactor and test the code (iterative!)
- Check with Pascal Analyzer or another tool
- Building the code
- Running all unit tests
- Generating the documentation
- Deploying to a target machine
- Performing a “smoke test” (just compile)

Let‘s practice

- 1
- 11
- 21
- 1211
- 111221
- 312211
- ??? Try to find the next pattern, look for a rule or logic behind !

Before R.

```
function runString(Vshow: string): string;
var i: byte;
Rword, tmpStr: string;
cntr, nCount: integer;
begin
cntr:=1; nCount:=0;
Rword:=''; //initialize
tmpStr:=Vshow; // input last result
for i:= 1 to length(tmpStr) do begin
  if i= length(tmpstr) then begin
     if (tmpStr[i-1]=tmpStr[i]) then cntr:= cntr +1;
     if cntr = 1 then  nCount:= cntr
     Rword:= Rword + intToStr(ncount) + tmpStr[i]
  end else
   if (tmpStr[i]=tmpStr[i+1]) then begin
      cntr:= cntr +1;
      nCount:= cntr;
   end else begin
     if cntr = 1 then cntr:=1 else cntr:=1; //reinit counter!
     Rword:= Rword + intToStr(ncount) + tmpStr[i] //+ last char(tmpStr)
end;
end; // end for loop
result:=Rword;
end;
```

After R.

```
function charCounter(instr: string): string;
var i, cntr: integer;  Rword: string;
begin
cntr:= 1;
Rword:=' ';
  for i:= 1 to length(instr) do begin
    //last number in line
    if i= length(instr) then
      concatChars()
    else
    if (instr[i]=instr[i+1]) then cntr:= cntr +1
    else begin
      concatChars()
      //reinit counter!
      cntr:= 1;
    end;
  end; //for
 result:= Rword;
end;
```

Refactoring Techniken

Einheit	Refactoring Funktion	Beschreibung
Package	Rename Package	Umbenennen eines Packages
Class	**Move Method**	Verschieben einer Methode
Class	**Extract Superclass**	Aus Methoden, Eigenschaften eine Oberklasse erzeugen und verwenden
Class	Introduce Parameter	Ersetzen eines Ausdrucks durch einen Methodenparameter
Class	**Extract Method**	Heraustrennen einer Codepassage
Interface	Extract Interface	Aus Methoden ein Interface erzeugen
Interface	Use Interface	Erzeuge Referenzen auf Klasse
Component	Replace Inheritance with Delegation	Ersetze vererbte Methoden durch Delegation in innere Klasse
Class	Encapsulate Fields	Getter- und Setter einbauen
Modell	Safe Delete	Löschen einer Klasse mit Referenzen

Metric based Refactoring

:ExtractMethod(EM)-MoveMethod(MM)-DataObject(DO)-ExtractClass(EC)

	EM	MM	DO	EC
• Normalized Cohesion	W	B	B	B
• Non-normalized Cohesion	W	B	B	B
• General Coupling	E	B	N	S
• Export Coupling	E	B	E	E
• Aggregated import coupling	B	W	W	W

  - Best, Worst, Expected, Suboptimal

Audits & Metric Links:

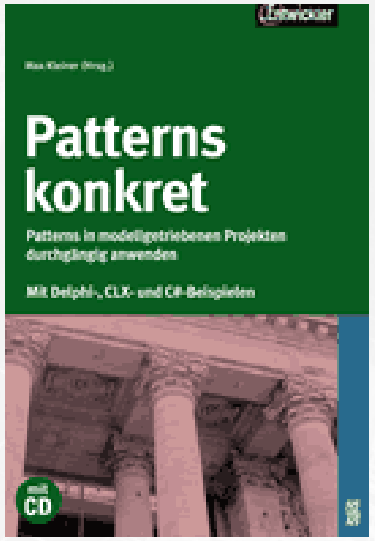
- Delphi XE Tool: Together

- http://www.modelmakertools.com/
- Report Pascal Analyzer: http://www.softwareschule.ch/download/pascal_analyzer.pdf
- *Refactoring* Martin Fowler (1999, Addison-Wesley)
- http://c2.com/cgi/wiki?CodeSmell
- Model View in Together: *www.softwareschule.ch/download/**delphi**2007_modelview.pdf*

Q&A

max@kleiner.com

www.softwareschule.ch



Entwickler

Max Kleiner (Hrsg.)

Patterns konkret

Patterns in modelligetriebenen Projekten durchgängig anwenden

Mit Delphi-, CLX- und C#-Beispielen

mit CD