Code Review Checklist

How far is a code review going?

“Metrics measure the design of code after it has been written, a Review proofs it and Refactoring improves code.”

The Problem is

- That the problem of source code smells is **additive** (new bugs arise but the old ones stay)
- Das kann bei uns nicht passieren! – wieso ist denn das passiert?
- Management is arrogance and developers are too proud to admit mistakes
- Programmers don't die, they just GOSUB without RETURN.

Improve the Situation Agenda

- Check your Documents with UML API?
- A Docu Structure to the 5 Top Level Metrics
- Back to Source Code Bugs & Smells Review
- Improve your Code (Metric Based Refactoring)
- Optimisation and Tools (Sonar & PA)

API Layers Structure & Scope

Metrics are for detecting bad code entities

- **Function** calls for type
- **Class** blocks for object
- **Module** components for file
- **Library** package for directory
- **Framework** deployment for device

API Layers to UML!

Metrics are for detecting bad code entities

- **Function** calls for Sequence Diagram
- **Class** blocks for Class Diagram
- **Module** components for Com.Diagram
- **Library** package for Package Diagram
- **Framework** deployment for Dep.D

API Layering Problems

Bad Coordination possible

- **Function** with Threads or threadsafe
- **Class** Access on objects with Null Pointer
- **Module** Bad Open/Close of Input/Output Streams or I/O Connections component
- **Package** return values or package namespace
- **Framework** in deployment code missing
- Docker Composer Access of **Container** image constructor on non initialized vars

Metrics deal with

Bad Structure (how cohesion & coupling)

- General Code Size (in module)
- Cohesion (in classes and inheritance)
- Complexity (>6!)
- Coupling (between classes or units)
  - Cyclic Dependency, Declare+Definition, ACD-Metric
- Interfaces or Packages (design & runtime)
- Static, Public, Private (inheritance or delegate)

Finally you can measure:

Bad Habit (no naming convention, no coverage)

Duplicated, dead code (side effects), bugs & smells

Long Methods (to much code), missing layering

Temporary Fields (confusion), comments or docs

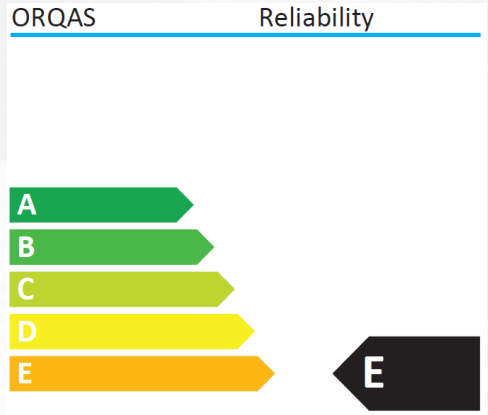
Long Parameter List (Object is missing)

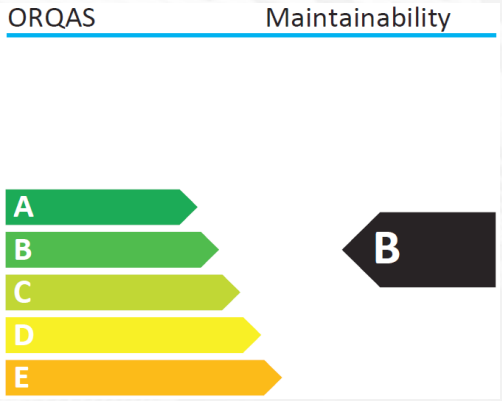
Data Classes (no methods)

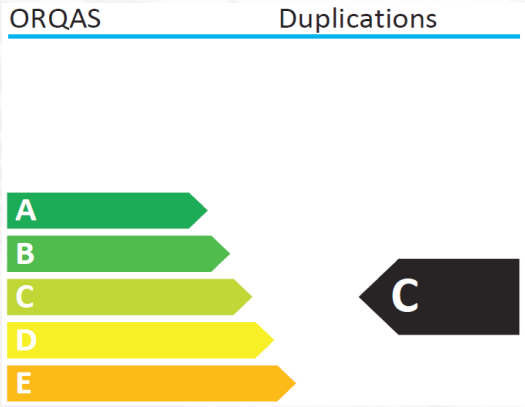
- Large Class with too many delegating methods

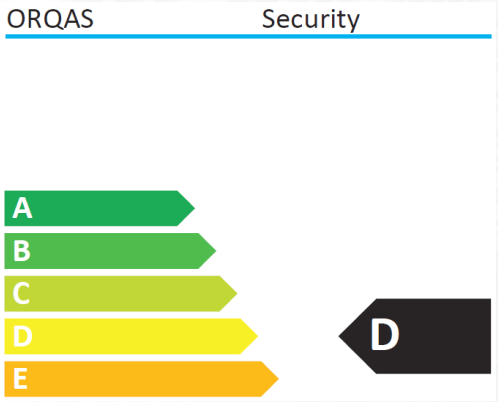
In a Kiviat Chart you get a Best Practices Circle!

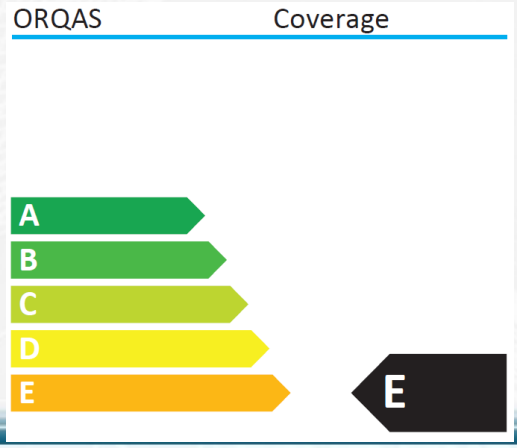
Review Doc Structure

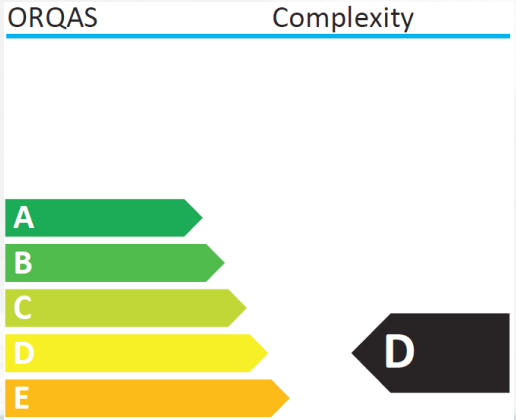






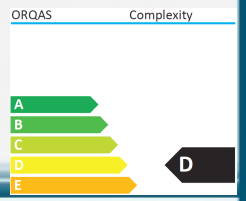






Review Doc Map

TIOBE Metrics	SONAR Rule Set	
1. Code coverage	[4]	Coverage
2. Abstract documentation	[6,8]	Size, Issues
3. Cyclomatic complexity	[7,3]	Complexity, Maintainability
4. Compiler warnings	[1]	Reliability
5. Coding standards	[1,3]	Reliability, Maintainability
6. Code duplication	[5]	Duplication
7. Fan out	[3,5]	Maintainability, Duplication
8. Security flaws	[2]	Security



https://www.tiobe.com/tqi/definition/
https://www.sonarqube.org/

Review Doc Structure II

- **Reliability** means less **Bugs**

The degree to which a system or component performs specified functions under specified conditions for a specified period of time.

- **Security** means less **Vulnerability**

The degree of protection of information and data so that unauthorized persons or systems cannot read or modify them and authorized persons or systems are not denied access to them.

- **Maintainability** means less **Code Smells**

The degree of effectiveness and efficiency with which the product can be modified in the sense of measured code smells which are non compliant to coding conventions or standards.

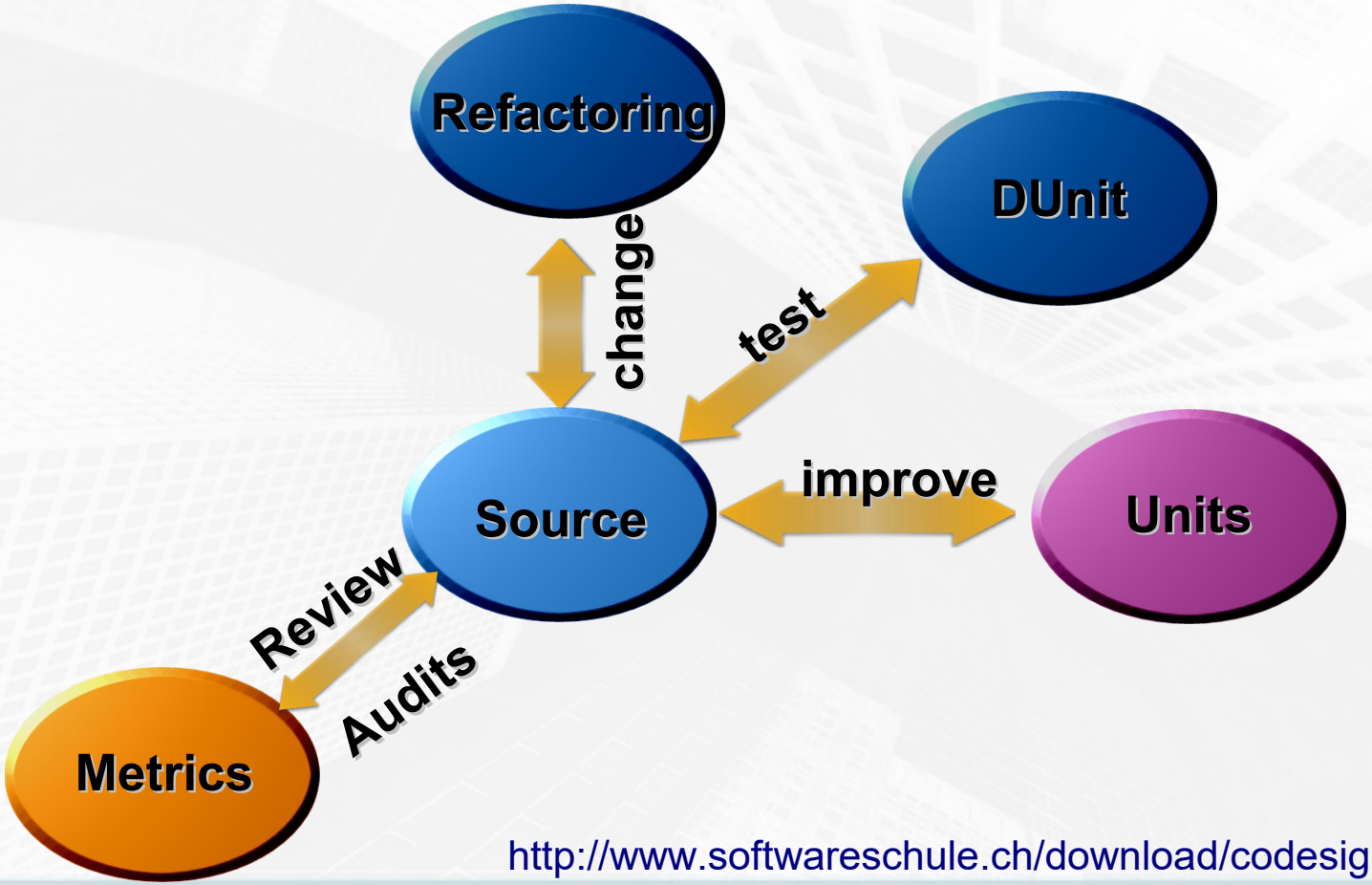
- **Operability means less Complexity**

The degree to which the product has attributes that enable it to be understood, learned, used and attractive to the user, when used under specified conditions.

- **Functionality means more Coverage and Size**

It measures the minimum number of test cases required for full test coverage.

Review Metric Context



http://www.softwareschule.ch/download/codesign_2015.pdf

When and why Metrics ?

Before 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 test scripts steps at a time
- Modify not only structure but also code format

Some kind of code smells

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

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 (a.b.m())
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

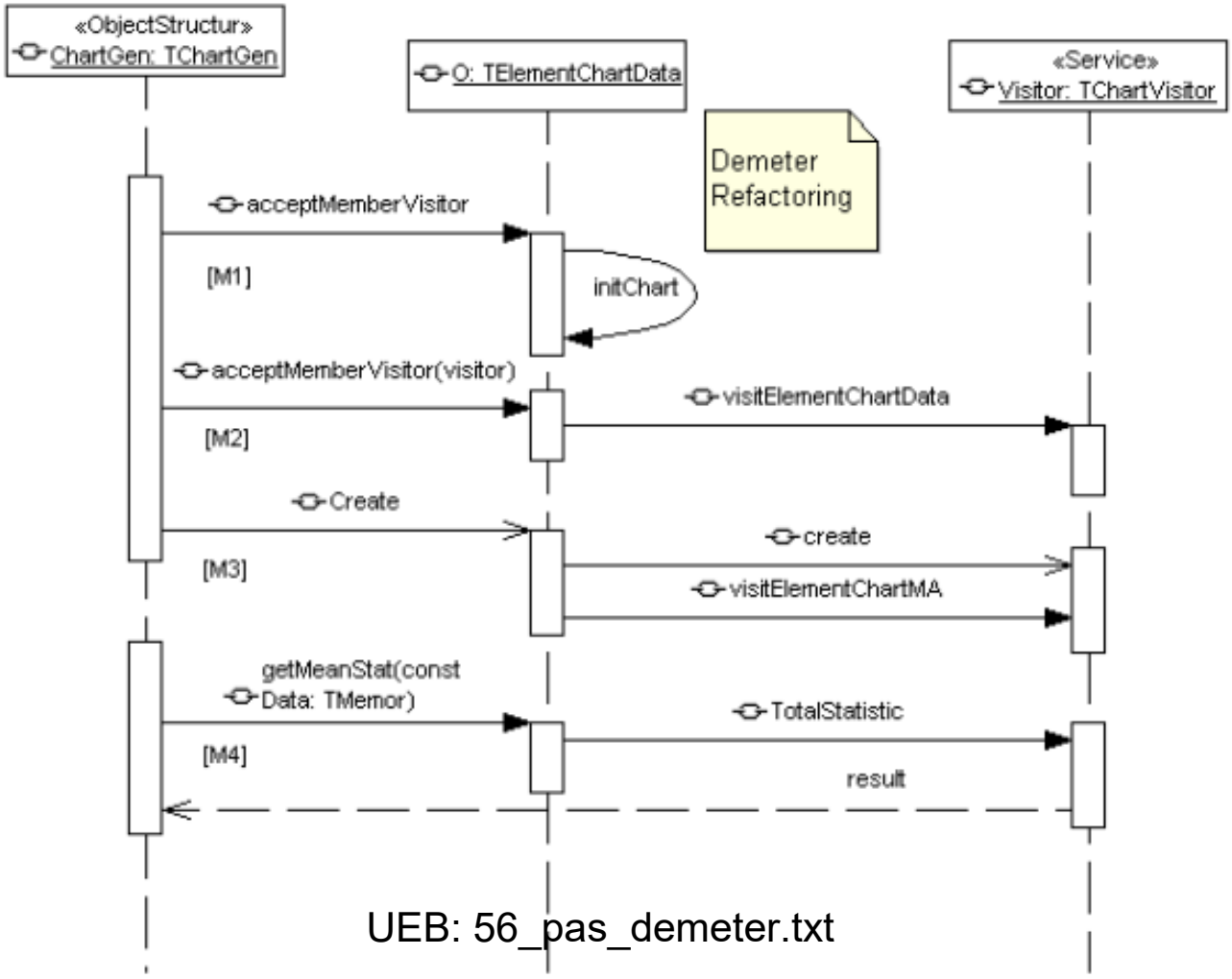
6. PUR (Package Usage Ratio) access information in a package from outside, see later
7. DD Dependency Dispersion (SS, Shotgun Surgery (Little changes distributed over too many objects or procedures → patterns missed))
8. CR Comment Relation
9. MDC (Module Design Complexity (Class with too many delegating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

Law of Demeter

```
Each unit should only talk to its friends; don't talk to strangers.
```

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

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;
```

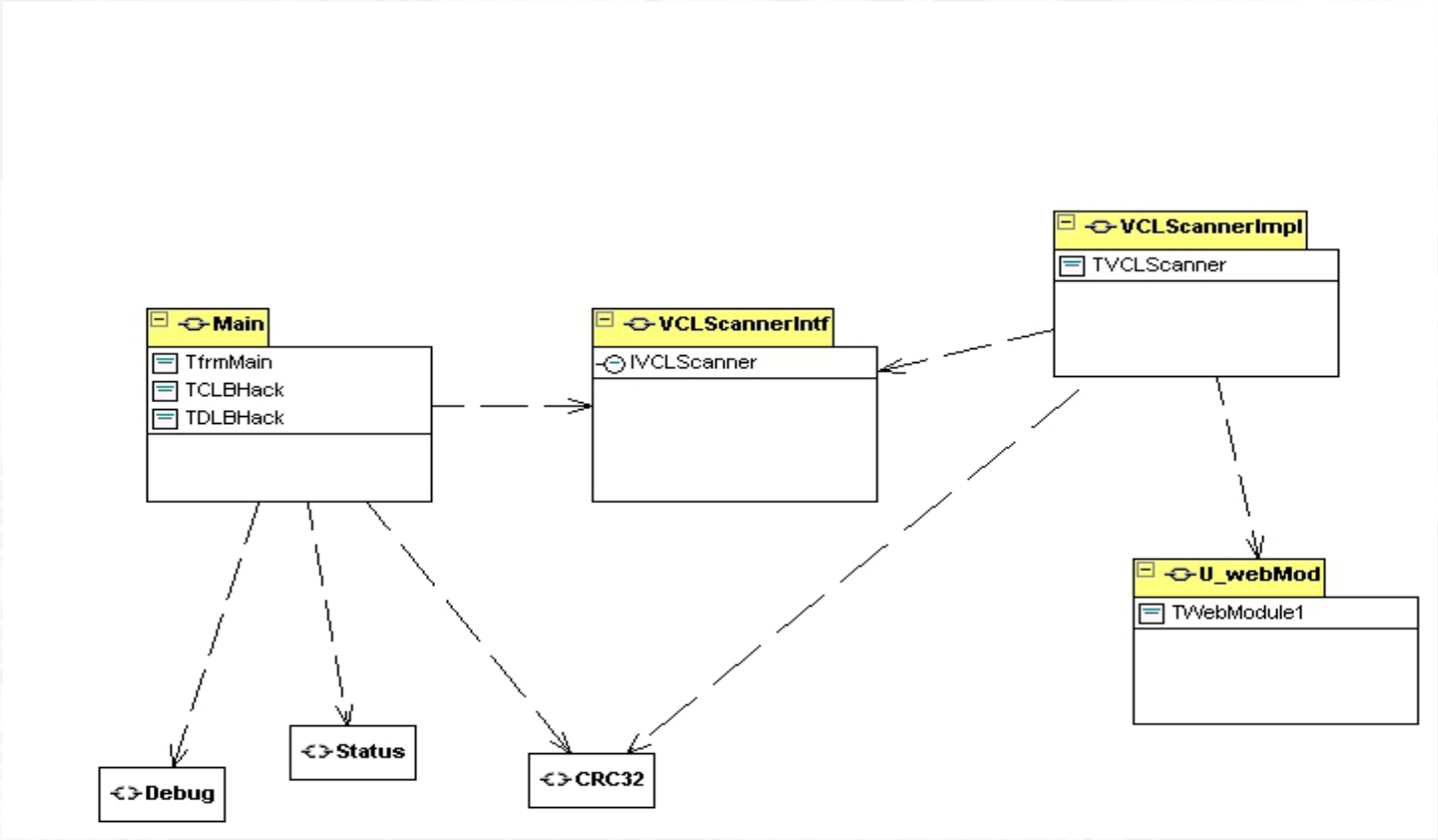
CC

- Check Complexity

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

Ueb: 164_code_reviews.txt

PUR Package Usage Ratio



DD – Test keywords knowledge

```
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;
```

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;  //cautious assign operator := or method D.assign()
```

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 (DUnit)
- Refactor and test the code (iterative!)
- Check with Analyzer or another tool
- Build 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 Methods

Einheit	Refactoring Funktion	Beschreibung
Package	Rename Package	Rename of Packages
Class	**Move Method**	Remove of Methods
Class	**Extract Superclass**	Aus Methoden, Eigenschaften eine Oberklasse erzeugen und verwenden
Class	Introduce Parameter	Replace of Expression w. Methodparameter
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- and Setter capsulate
Model	Safe Delete	Delete a Class with References

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:

- https://www.sonarqube.org/
- http://www.softwareschule.ch/

- Report Pascal Analyzer: http://www.softwareschule.ch/download/pascal_analyzer.pdf
- Refactoring Martin Fowler (1999, Addison-Wesley)
- http://c2.com/cgi/wiki?CodeSmell

- https://www.slideshare.net/maxkleiner1/code-review-with-sonar
- Model View in Together:
- www.softwareschule.ch/download/delphi2007_modelview.pdf

Q&A

max@kleiner.com

www.softwareschule.ch/maxbox