

Code Reviews planen und durchführen

Ein Code Review ist nach gewissen Qualitätsregeln (Audits) und Metriken durchführbar (Struktur vor Funktion). Ein Review basiert auf Standards und Standards (Muster) fördern die Simplifikation im Softwarebau.

Kleiner
K o m m u n i k a t i o n

Code Review Ziel

- Das Ziel eines Code Review besteht darin, ein klares **Verständnis** darüber zu erhalten, wie ein Software-System strukturiert ist, welche Protokolle, Formate und Schnittstellen anhand der Architektur vorgesehen sind; Eine Architektur basiert auf: Plattform, Framework, Topologie
- Ein Code Review prüft und beurteilt die nichtfunktionalen Anforderungen, Qualität (Fehlerminimum) bezüglich Architektur und Substanzwert !
- Eine Inspektion spezialisiert sich auf die Fehlerfindung, nicht ihrer Beseitigung. Generelle Reviews prüfen auch den Projektfortschritt.

Nichtfunktionale Anforderungen

- Modularisierung
- Erweiterbarkeit, Flexibilität
- Wartbarkeit, Wiederverwendung
- Testbarkeit der Spezifikation
- Software-Qualität, Sicherheit und Robustheit

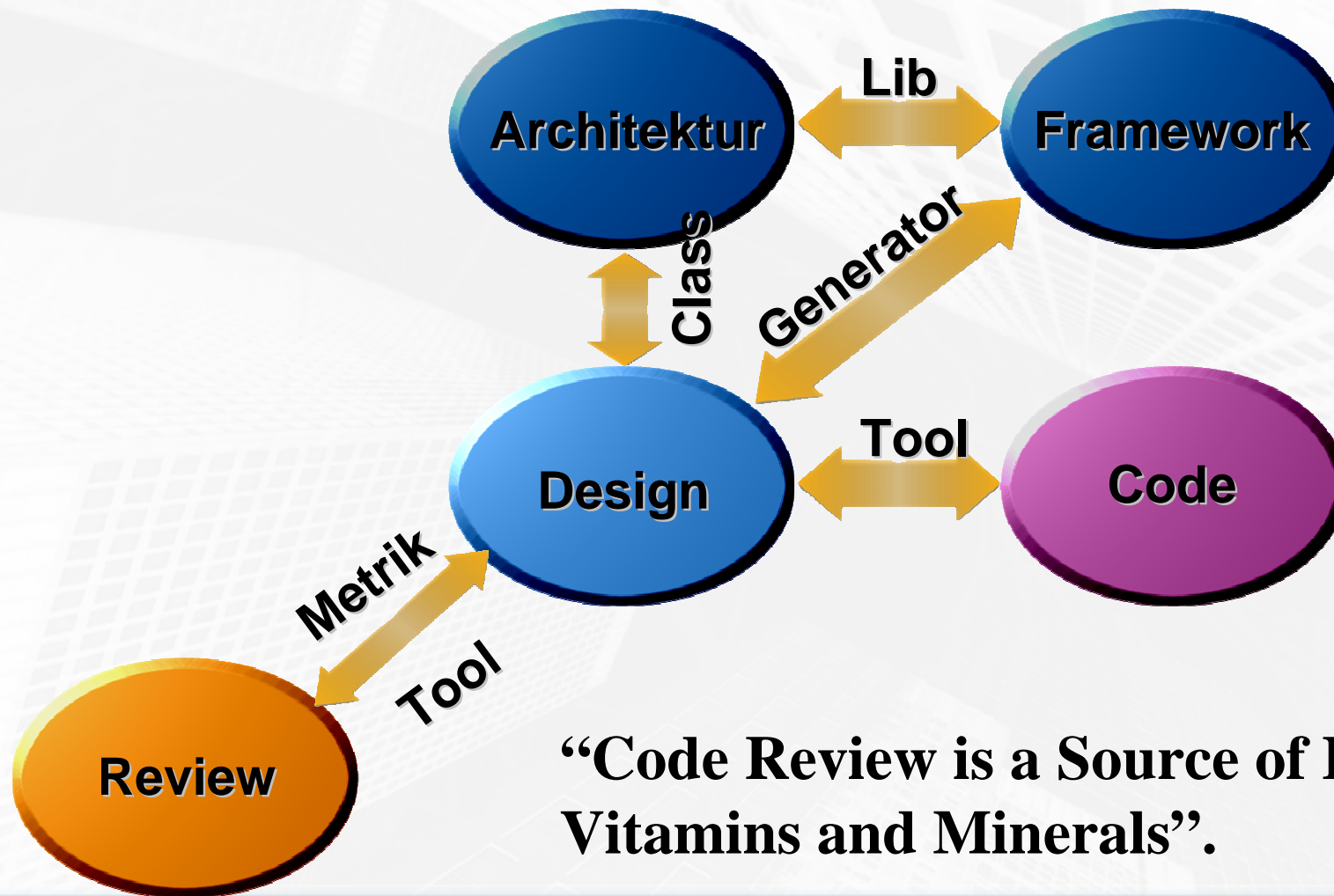
Sekundäre Anforderungen: (Wechselwirkungen)

- Performance
- Skalierbarkeit
- Redundanz und Verfügbarkeit
- Portabilität, Mehrsprachigkeit

Wie misst man T. Anforderungen?

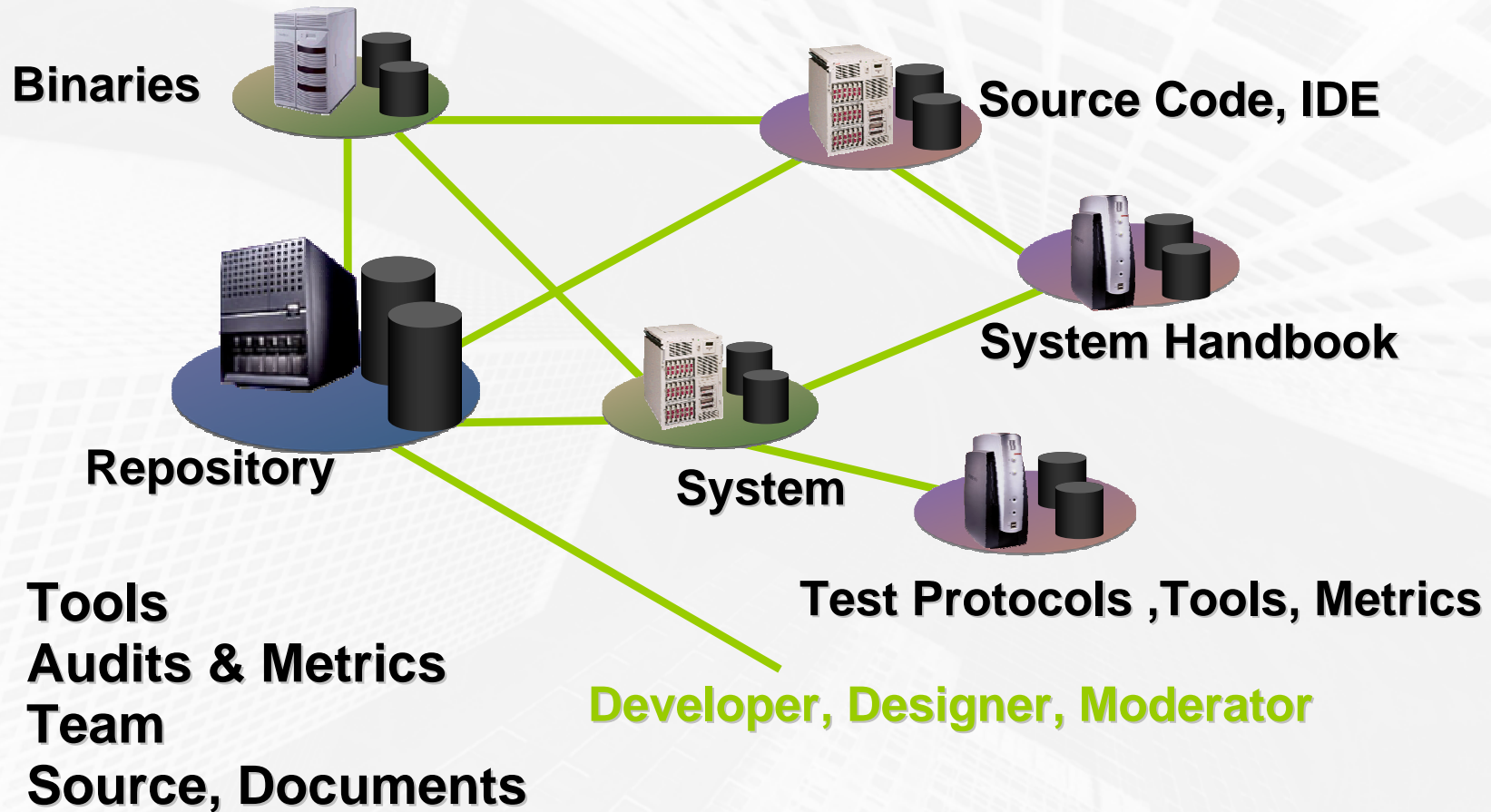
- General Code Smoke Test (compilation)
- Comments, Coding Conventions (D. by Contract)
 - Type safety, Declaration, RTTI, Thread Safeness
- Error Handling, Resource Leaks (memory)
- Layers, Tiers and Control Structures (run time)
- Functions and Bindings (design time)
- Bug Fixes & Release Planning
- Versioning & System Handbook

Review Umfeld



“Code Review is a Source of Essential Vitamins and Minerals”.

Review Elemente



Delphi Review Tools ;)

- Pascal Analyzer (4.0.1) www.peganza.com
- eyebol (1.30d) is a Delphi code analyzer, refactoring and spell checking tool for your projects. www.eyebol.com
- Together Audits & Metrics
- ModelMaker www.modelmakertools.com

Sprache und Tools

Statische Typisierung entlastet jeden Review mit den Vorteilen:

- Optimaler Speicherverbrauch und Zugriff, Methoden-Dispatch zur design time
- Compiler-Prüfungen durch Typsicherheit
- IDE Support durch Typinformation
- Bessere Kommunikation im Code durch explizite Angabe der Methodensignatur

Vorbedingungen

- Dokus, Testklassen zu Algorithmen
- Vorabversionen und Updates
- Installation, Support, Hotline der Firma bekannt
- Volle Verfügbarkeit über den Sourcecode
- Compilationsfähigkeit erstellt (\$D, \$L, tdb32info)
- Benchmarks und Memoryleaks geprüft
- Tools und Review Kriterien vorhanden
- Sprache und Form bei Dokumenten festlegen
- Referenzarchitektur und Patterns bekannt

Modularisierung

- Patterns Suche, Test mit einem Dependency Analyzer
Suchen nach einem Schichtenmodell (MVC etc.)
 - Ist das Programmdesign optimal gewählt?
 - Zu viele globale Variablen?
 - Schlecht aufgeteilte Methoden bzw. Klassen?
 - Redundante Codesequenzen? (don't repeat yourself)
- Modularisierung (zur design- oder runtime ?)

```
TWebModule1 = class(TWebModule)
```

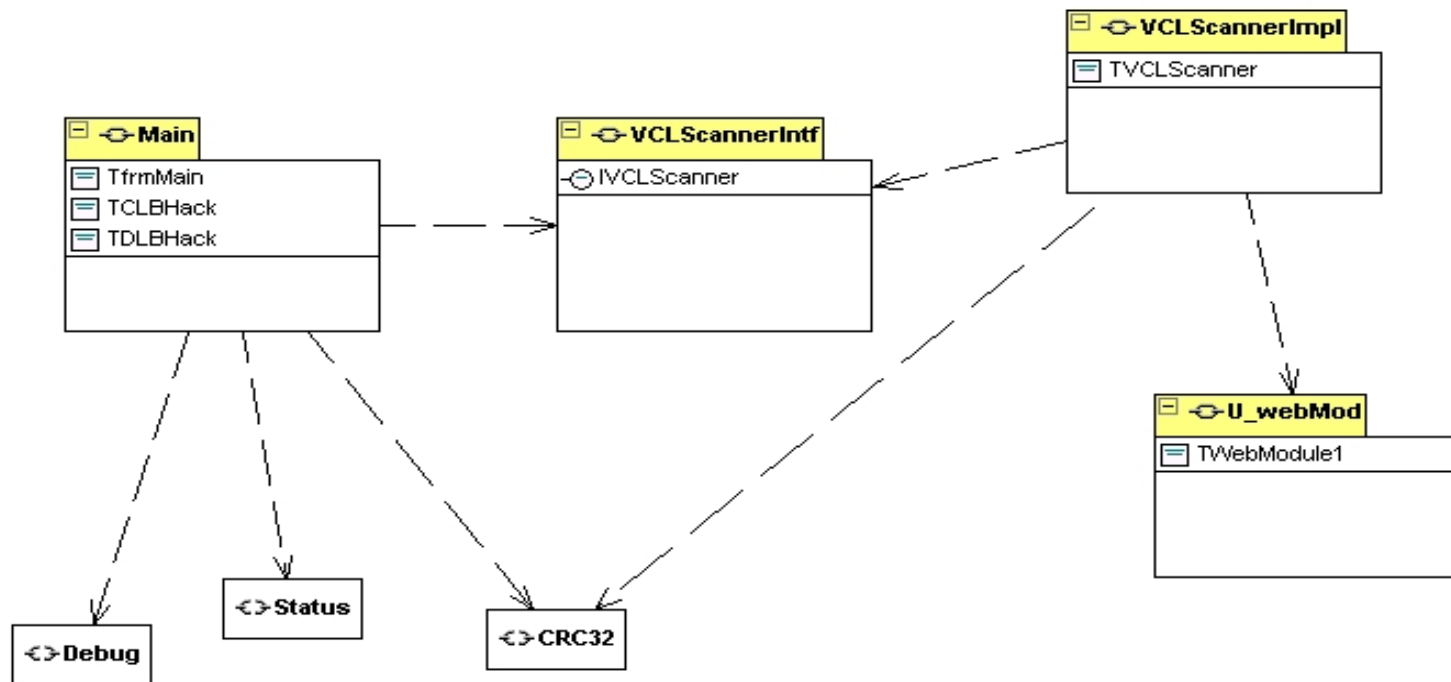
```
    HTTPSoapDispatcher1: THTTPSoapDispatcher;
```

```
    HTTPSoapPascalInvoker1: THTTPSoapPascalInvoker;
```

```
    WSDLHTMLPublish1: TWSDLHTMLPublish;
```

```
    DataSetTableProducer1: TDataSetTableProducer;
```

Module mit einem PAC prüfen



Erweiterbarkeit

Sind Interfaces, Packages, Namespaces vorhanden?

type

```
{Invokable interfaces must derive from IInvokable }
```

```
IVCLScanner = Interface(IInvokable)
```

```
['{8FFBAA56-B4C2-4A32-924D-B3D3DE2C4EFF}']
```

```
function PostData(const UserData : WideString; const  
    CheckSum : DWORD) : Boolean; stdcall;  
procedure PostUser(const Email, FirstName,  
    LastName : WideString); stdcall;
```

Erweiterbarkeit II

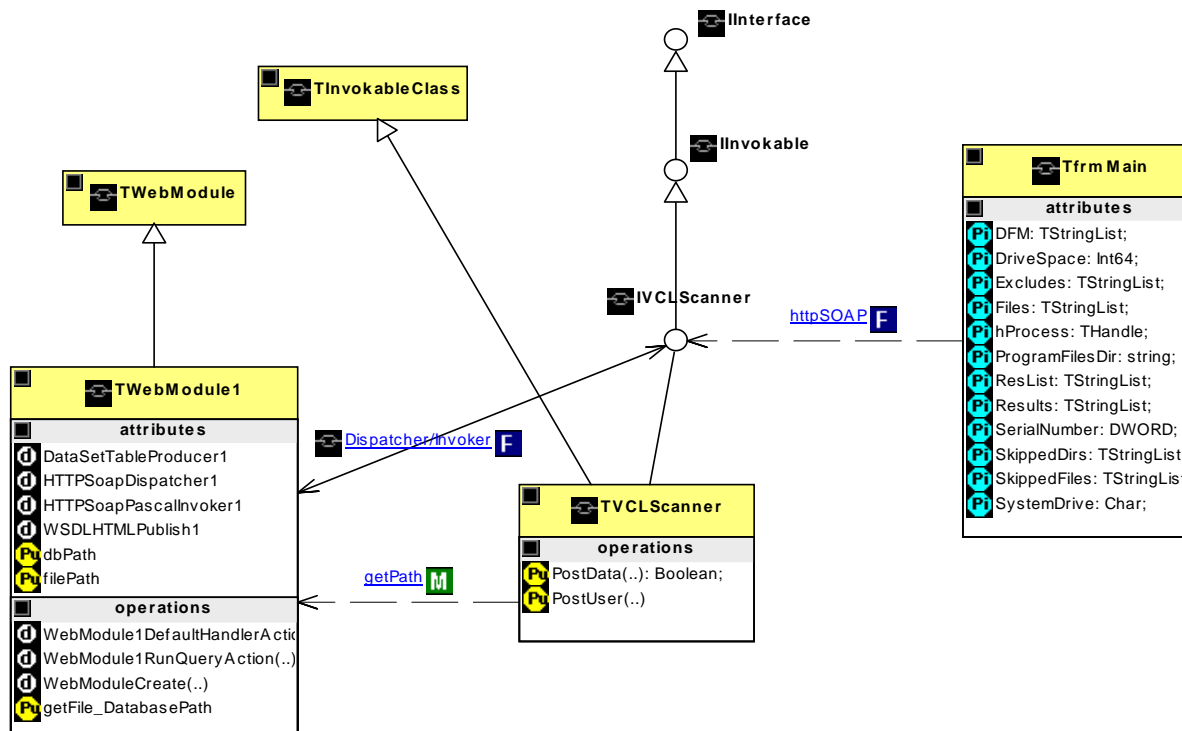
Sind Business Objekte vorhanden (Ausbaufähig)?

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

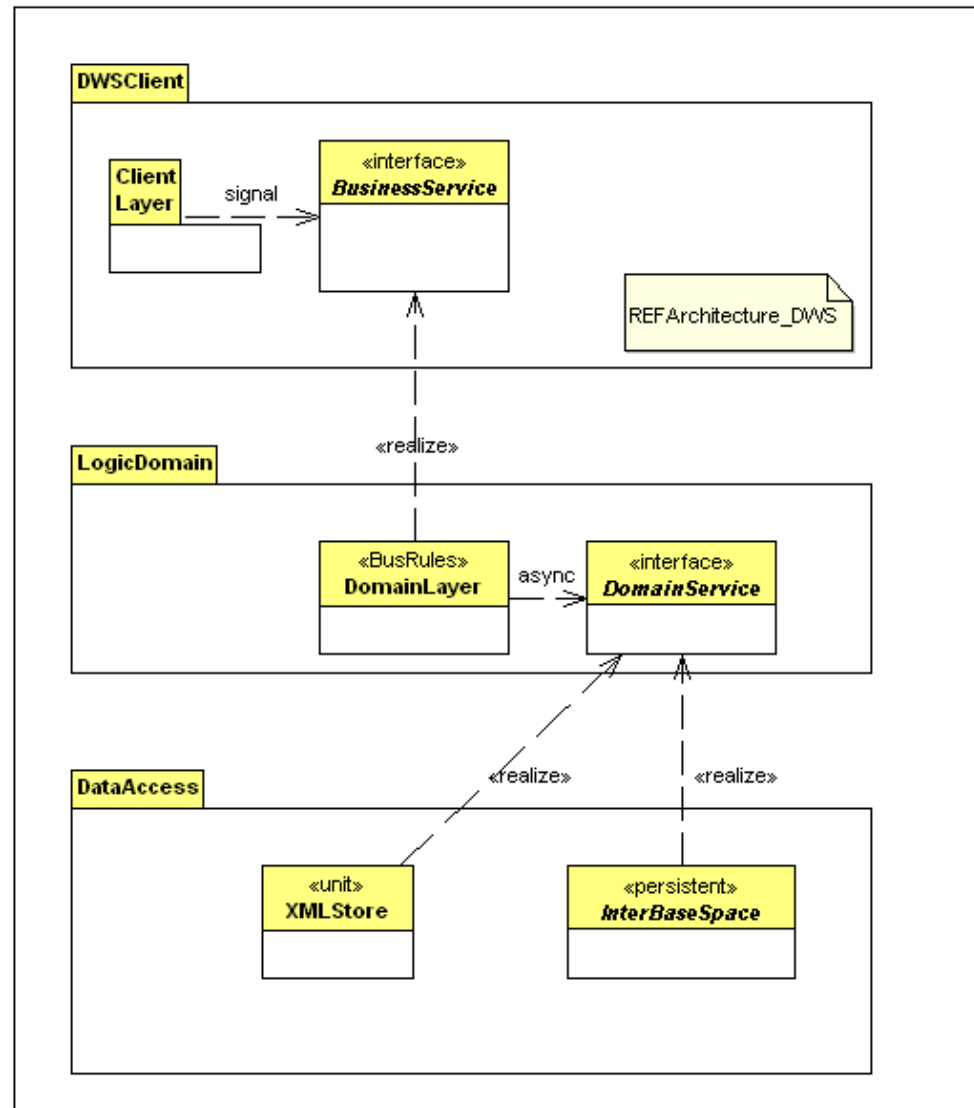
1. The Business-Class inherits from a Data-Provider
2. The query is part of the class
3. A calculation or business-rule has to be done
4. The object is independent from the GUI, GUI only calls the dependent object

“Business objects are sometimes referred to as conceptual objects, because they provide services which meet business requirements”

Optischer Review eines Class Diagram



Flexibilität Mit Dependency Inversion:



Anti Flexibilität ?

Mit TSQLConnection Bsp.

- Connection:= TSQLConnection.Create(NIL);
- with Connection do begin
- ConnectionName:= 'VCLScanner';
- DriverName:= 'INTERBASE';
- LibraryName:= 'dbexpint.dll';
- VendorLib:= 'GDS32.DLL';
- GetDriverFunc:= 'getSQLDriverINTERBASE';
- Params.Add(,user_name=SYSDBA');
- Params.Add('Password=masterkey');
- with TWebModule1.create(NIL) do begin
- getFile_DataBasePath;
- Params.Add(dbPath);

Wartbarkeit

- Namenskonvention, Namensräume, Kommentare
- Bridge zwischen Interface und Implementation (siehe auch Modularisierung)
- Installation und Configuration Guide durchsehen
- Abhängigkeiten der Module klären:

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

Wartbarkeit II

- die Dokumentation, insbesondere die exakte Spezifikation von Schnittstellen (Interfaces)
- die lokale Verständlichkeit von Anweisungen resp. von Kommentar im Code, der Parameter
- das Vermeiden von Duplicated-, Dead Code
- das Vermeiden globaler Variablen
- in das Programm eingebaute Prüfungen der Annahmen, die man über Zustände hat (Assertions)
- ein möglichst großer Umfang von automatisch ausführbaren Tests für das System (DUnit)

Wartbarkeit III

- die Dokumentation, insbesondere die Verständlichkeit von Modulen (Komponenten)
- Program MailingLabels //pseudo code dokumentieren und referenzieren!

Begin

ask the user for the name of the list

read that list into memory

ask the user how the list should be sorted: name or zip code

sort the list

ask the user about where to send the output: screen or printer

print the labels

end. { Program MailingLabels }

Compilerdirektiven warten

- Die Direktiven sollten als *.inc file vorhanden sein ({ $\$I$ IdCompilerDirectivs.inc}, { $\$I$ IdVers.inc}).

{ $\$H$ -}

STRING Längenbyte + ARRAY[1..255] OF CHAR;

STRING[16]Längenbyte + ARRAY[1..15] OF CHAR;

{ $\$H$ +}

STRING Zeiger auf: ARRAY[0..?] OF CHAR + #0;

STRING[16]Längenbyte + ARRAY[1..15] OF CHAR;

Testbarkeit Praxis

```
function letAllowInstances(const MainForm, cmdLine:
                          PAnsiChar): boolean;
var keyWindow: HWND;
begin
  result:= false;
  keyWindow:= FindWindow(MainForm, NIL);
  if (keyWindow <> 0) and not
      FindCmdLineSwitch(cmdLine, true) then
    //set existing instance
    ShowWindow(keyWindow, SW_SHOWNORMAL)
  else
    result:= true;
end;
```

Testbarkeit Code

- Exception handling

```
{$IFDEF DEBUG}
```

```
Application.OnException:= AppOnException;
```

```
{$ENDIF}
```

- Assert function

```
accObj:= TAccount.createAccount(FCustNo,  
                                std_account);
```

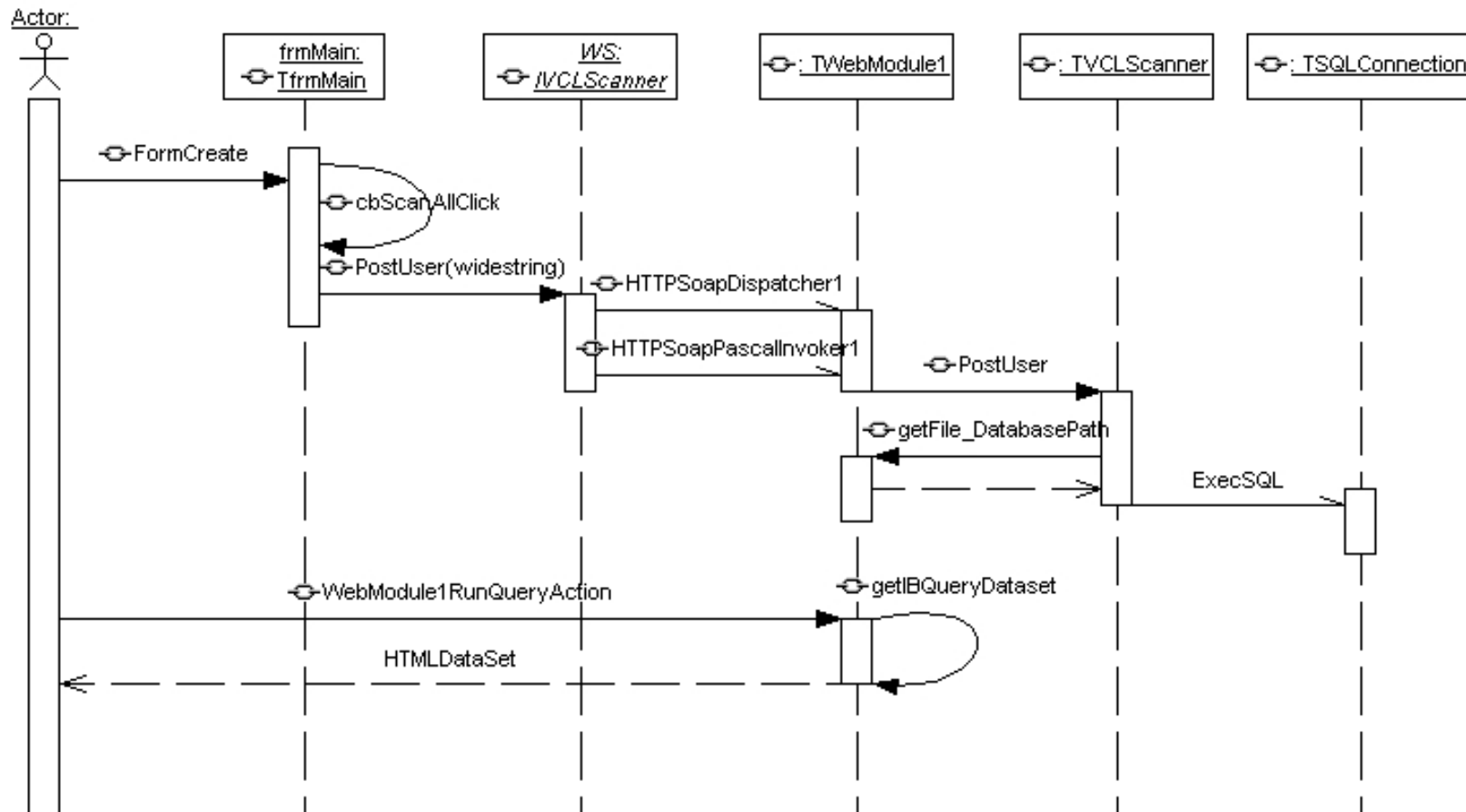
```
assert(aTrans.checkOBJ(accObj), 'bad OBJ cond.');
```

- Logger

```
LogEvent('OnDataChange', Sender as TComponent);
```

```
LogEvent('BeforeOpen', DataSet);
```

Testen anhand eines SEQ



Software Sicherheit

- The Delphi compiler hides the fact that the string variable is a heap pointer to the structure but setting the memory in advance is advisable:
- Check against Buffer overflow or TryIntToStr

```
path: string;
```

```
setLength(path, 1025);
```

```
var Source, Dest: PChar;
```

```
begin
```

```
    Source:= aSource;
```

```
    Dest:= @FData[FBufferEnd];
```

```
    if BufferWriteSize < Count then
```

```
        raise EFIFOStream.Create('Buffer over-run.');
```

Software Sicherheit II

- Avoid pointers as you can
- Ex. of the win32API:

```
pVinfo = ^TVinfo;  
function TForm1.getvollInfo(const aDrive: pchar; info: pVinfo): boolean;
```

```
//refactoring from pointer to reference (const pointer with automatic deref)  
function TReviews.getvollInfo(const aDrive: pchar; var info: TVinfo):  
    boolean;
```

“Each pointer or reference should be checked to see if it is null. An error or an exception should occur if a parameter is invalid.”

Software Sicherheit III

- Exception Handling und Range Checking prüfen !

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

```
{ $R+ }  
  SetLength(Arr,2);  
  Arr[1]:= 123;  
  Arr[2]:= 234;  
  Arr[3]:= 345;  
{ $R- }
```

Performance (ohne Profiler)

Beim **Optimisation Report** will man mögliche Zeitlöcher vermeiden, vor allem im Bereich der Referenzübergabe oder dem inlining / including

Missing “const” for unmodified string parameter

```
procedure MyProc(S: string);  
begin
```

```
...
```

```
if S = 'Foo' then begin // !! S is not changed
```

- Are you using blocking system calls when performance is involved, e.g inlines?
- Consider using a different thread for code making a function call that blocks.
- Are whole objects duplicated but only ref. are needed?

Performance (Effizienz)

- Da es nicht sinnvoll ist, einen Algorithmus nur bzgl. einer einzigen Eingabe zu untersuchen, betrachtet man seine Effizienz bzgl. aller Eingaben.
- Da die Laufzeit und der Platzbedarf mit der Größe der Eingabe erwartungsgemäß zunimmt, misst man die Effizienz meist bzgl. der Eingabegröße. Die Effizienz ist damit eine Funktion, deren Parameter die Eingabegröße ist.

Skalierbarkeit

Bei Parametrisierung und Speicherbedarf will man mögliche Speicherlöcher vermeiden, (try/finally Test)

```
Buffer: PChar; //not a buffer before you use getMem
```

```
Size:= FileSize(F);
```

```
GetMem(Buffer, Size); //allocates n-Bytes on the heap
```

```
BlockRead(F, Buffer^, Size);
```

- All allocated memory needs to be freed when no longer needed. Make sure memory is released in all code paths, especially in error code paths.
- Are all objects (Database connections, Sockets, Files, etc.) freed even when an error occurs? → procexp.exe

Portabilität

Does the code re-write functionality that could be achieved by using an existing API?

```
function MakeTempFilename: string;
begin
  {$IFDEF LINUX}
    Result := tempnam(nil, 'Indy'); {do not localize}
  {$ELSE}
    SetLength(Result, MAX_PATH + 1);
    GetTempFileName(PChar(ATempPath), 'Indy', 0, PChar(result));
    Result := PChar(Result);
  {$ENDIF}
end;
```

Portabilität ?

- Steht hingegen ein eigenes Framework im Vordergrund, das mit den Unterklassen flexibel und skalierbar auf Änderungen hin reagieren muss, sollte man das Subclassing mit Vererbung vorziehen.

Konkret: Beim Komponentenbau spielt die Vererbung die erste Geige, beim späterem Einsatz oder Integration einer Komponente ist Aggregation oder Komposition gefragt, welche in den meisten Fällen der Vererbung vorzuziehen ist.

QS Review Checklist

1. **Standards** - are the Pascal software standards for name conventions being followed?
2. **Bugs** - Are the changes generally correct?
3. **Are the Requirements Well Understood (Multilang)?**
4. **Are all program headers completed?**
5. **Are code changes commented appropriately?**
6. **Does documentation use Correct Grammar?**
7. **Are release notes Clear? Complete?**
8. **Installation Issues, Licenses, Certs. Are there any?**
9. **Version Control, Are output products clear?**
10. **Test Instructions - Are they any? Complete?**

Fehleranalyse und Review Relevanz

FehlerTyp-Nr / Bezeichnung / Verteilung / Review relevant

10 Dokumentation - Kommentare, 0.5, Ja

20 Syntax - Syntax-Fehler (vor dem Kompilieren), 49.3, Nein

30 Build - package library, version control, 15.4, Ja

40 Assignment - Dekl., Doppelte Bezeichnungen, 8.8, Ja

50 Interface - Prozedur-Aufrufe und Referenzen, I/O, 1.5, Ja

60 Logging - Fehler-Nachrichten, Checks, 2.2, Ja

70 Daten - Struktur, Inhalt, Format, 1.5, Ja

80 Funktion - Schleifen, Rekursionen, Logik, 16.6, Nein

90 System - Konfiguration 1.5, Ja

100 Umgebung - Absturz, Compiler, Support, 3.7, Ja

Code Review umsetzen

“The code reviewer must produce a list of all major areas that need to be addressed by the developer. These should be commented into the Audit Report after the review as a statement of the hard work done by the reviewer and as a open item list (OIL) of addressable items to do for the developer.”

Fragen und hoffentlich Antworten ?

max@kleiner.com

