IT WORKS – WE CARE
TECHTALK

Herzlich Willkommen

Paul Rohorzka

IT WORKS – WE CARE
TECHTALK

Pragmatisches Testen von Access-Anwendungen

Teil 1

Präsentation v1.4 für Download als PDF

Agenda Teil 1

- Einführung in das Testen von Software
- AccUnit und SimplyVbUnit
- Unit Testing: Dos, Don'ts & Tipps
- Test-First Development

Einführung in das Testen von Software

Was es gibt und wie es geht

Regression durch Bugfix

oder

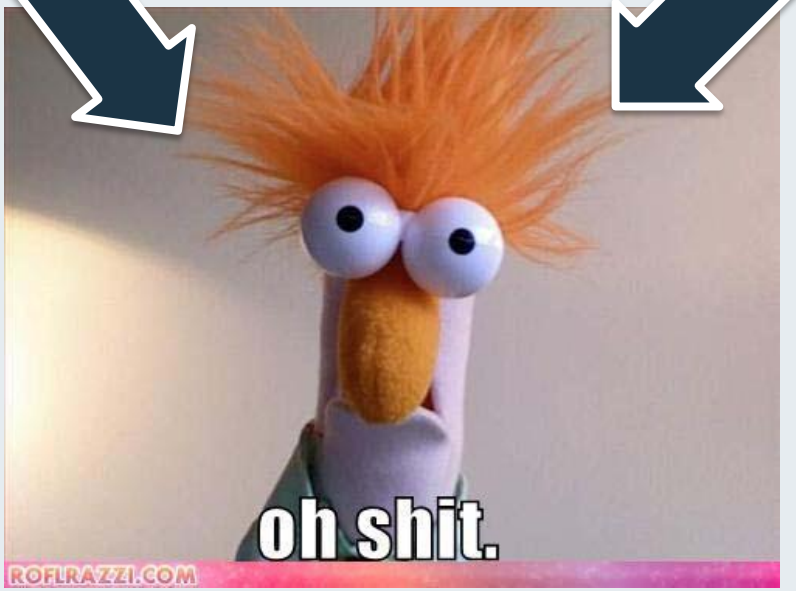
Von offensichtlichen Lösungen und verborgenen Katastrophen

Demo

Was wir gelernt haben

Wir wissen nicht immer genau welche **Auswirkungen** eine **Änderung** nach sich zieht.

Wenn wir nicht **testen**, zeigen sich **Fehler** oft erst **zu spät**.

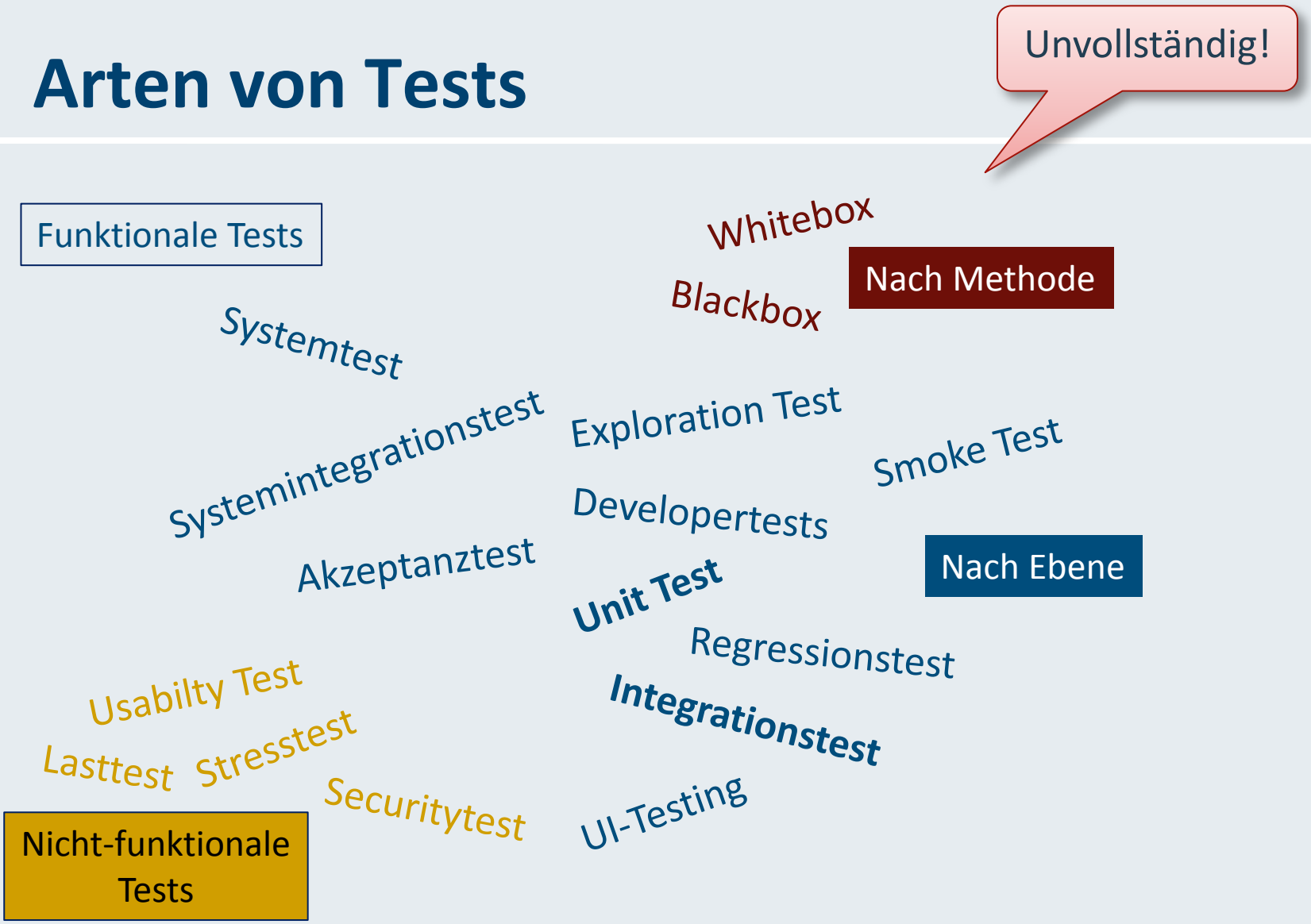


IT WORKS – WE CARE
TECHTALK

Warum testen?

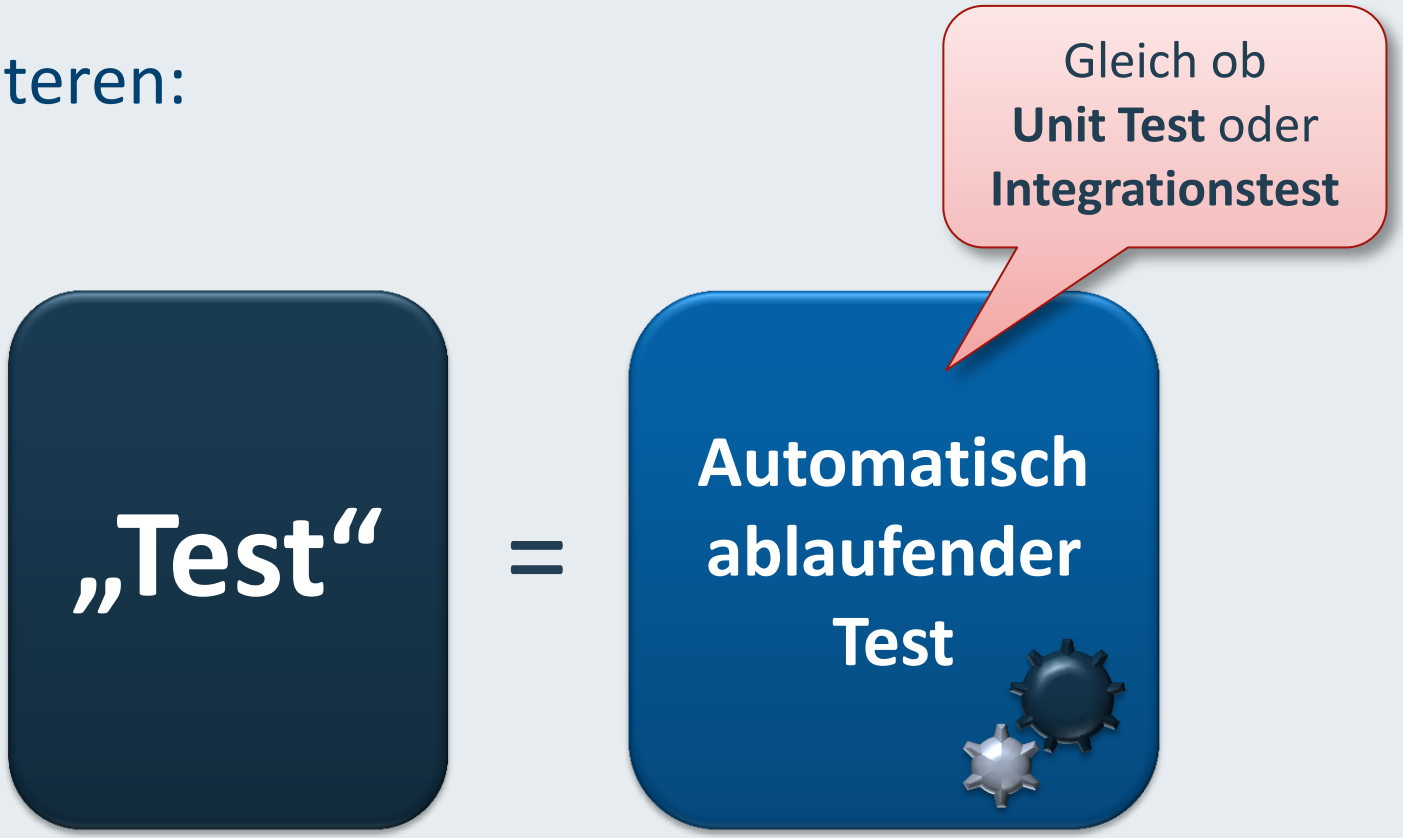
- Sicherstellen der **Funktionalität** getesteter Bereiche entsprechend einer **Spezifikation**
- Erkennen von **Auswirkungen** auch in "entfernten" Teilen der Applikation
- Erhöhte **Qualität**
  - Weniger Bugs in Produktion
- Erhöhte **Effizienz**
  - Change- & Feature-Requests sowie Bug Fixes können in stabiler Qualität umgesetzt werden.

Arten von Tests



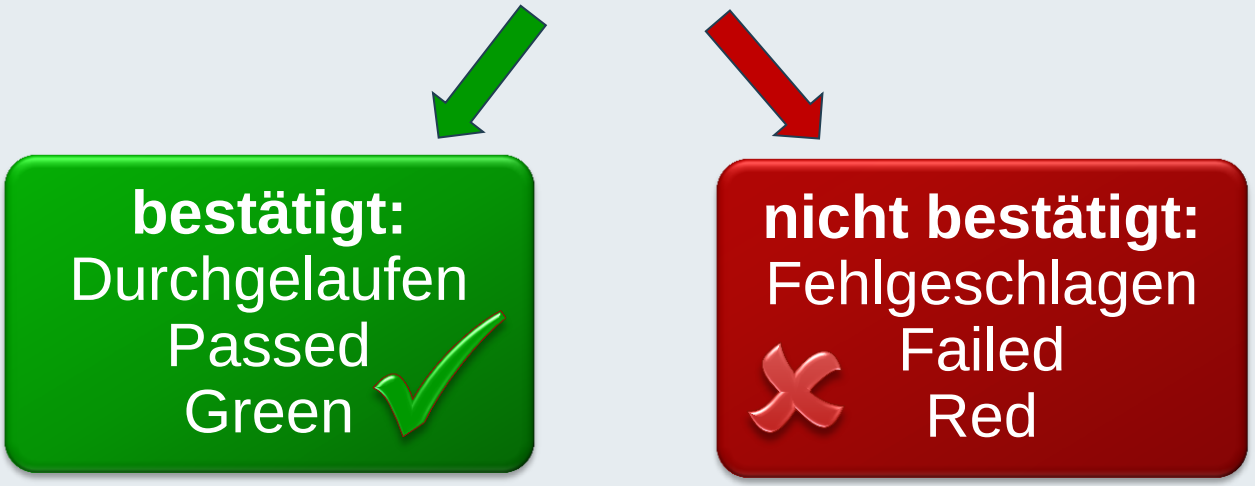
Begriffsklärung

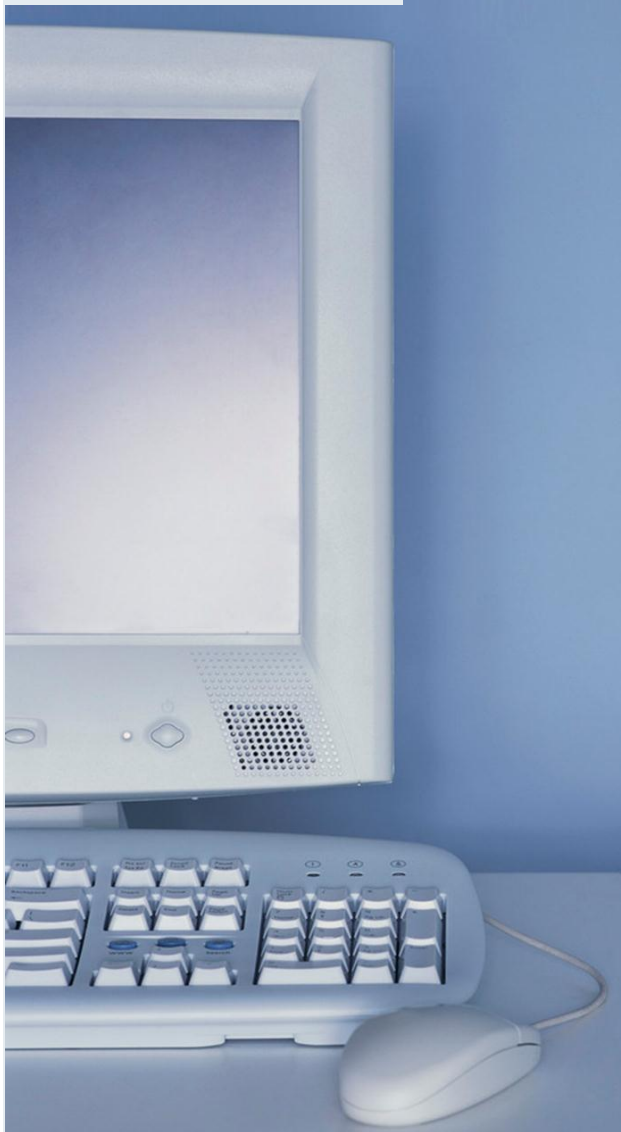
Im Weiteren:



Was ist ein Unit Test?

Ein Unit Test ist
ein Stück **Code**
das **automatisiert**
anderen Code aufruft und
danach **Behauptungen** überprüft.





Verhinderte Regression durch Tests

oder

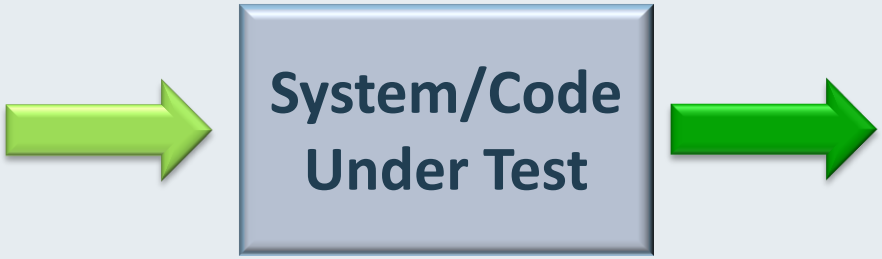
Wie schön nicht alles hätte sein können

Demo

Wichtiger Grundsatz

Testen ≠ Ausprobieren

Ich kann nur dann etwas **testen**,
wenn mir **klar** ist,
welche **Eingaben**
welches **Verhalten**
bewirken sollen.



AccUnit und SimplyVbUnit

Tool für Unit Testing in Access-Anwendungen

Erstellen von Unit Tests mit AccUnit

Demo

AccUnit und SimplyVbUnit

- OpenSource
- SimplyVbUnit (SVU) von Kelly Ethridge
  http://simplyvbunit.sourceforge.net
  - Unit Testing Framework für VB6
  - Geschrieben in VB6
  - Constraint based assertions ~NUnit 2.4 (Fluent API)
- AccUnit von Josef Pötzl (und Paul Rohorzka)
  http://wiki.access-codelib.net/**a**cc**u**nit
  - Integration von SimplyVbUnit in VBA
  - COM-AddIn für den VBA-Editor
  - Geschrieben in C# 2010

Tests schreiben

- Tests sind in **Testfixtures** (Klassen) definiert
  - Können durch AccUnit eingefügt werden
- Jede **öffentliche Sub-Methode** ist ein Test
  - Ohne Parameter (Ausnahme RowTest)
- Aufbau eines Tests:
  - **A**rrange – die zu testende Situation herstellen
  - **A**ct – den zu testenden Code ausführen
  - **A**ssert – die Annahmen überprüfen

Asserts

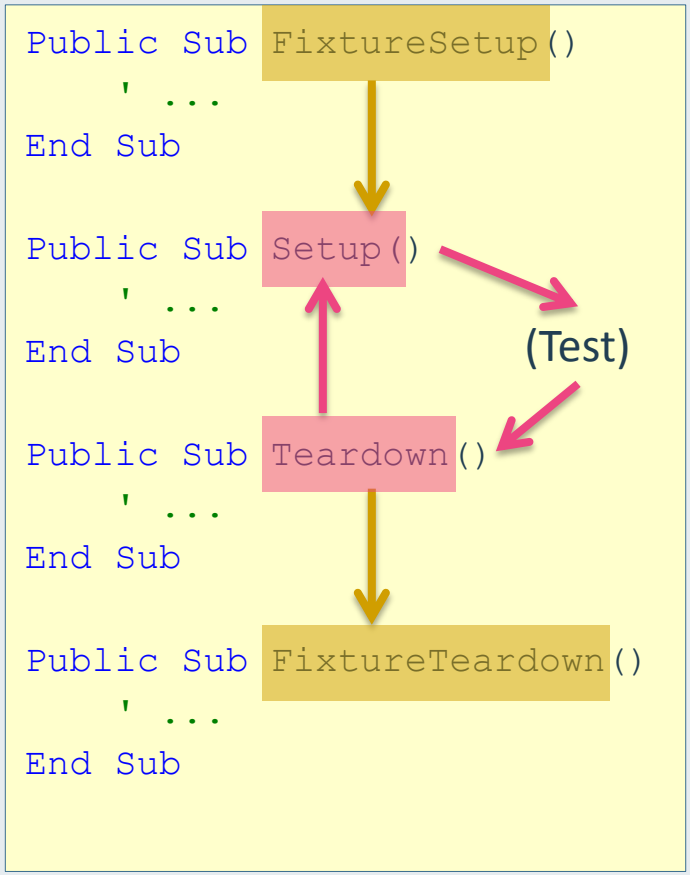
- Ein Assert ist eine Aussage über den **erwarteten Zustand** des Testobjekts nachdem der CUT gelaufen ist.
- Wenn der Zustand nicht entspricht, ist der Test fehlgeschlagen.
- Asserts werden **deklarativ** formuliert
- Oft ist ein Assert ein Vergleich
  - des tatsächlichen Wert (actual)
  - mit einem erwarteten Wert (expected)

Beispiele für Asserts in SimplyVbUnit

```
Assert.IsTrue ProcessSucceeded
Assert.IsNotNull CustomerFromDatabase
Assert.Throws ExpectedErrorNumber, , _
              ExpectedErrorMessage
Assert.That Actual, Iz.EqualTo(expected).Within(0.001)
Assert.That Result, Iz.Not.EqualTo(SomeEvilValue)
Assert.That Result, Iz.AtLeast(LowerBound)
Assert.That Numbers, Iz.All.LessThan(UpperBound)
Assert.That Elements, Has.All.Type(vbInteger)
Assert.That CityInfos, Has.Some.Contains("Wien")
Assert.That Number, Iz.InRange(0, 100) _
                    .OrElse.EqualTo(-1)
```

Testlifecycle

- Vor- und nach jeder Testmethode und -klasse kann Code ausgeführt werden
- Setup/Teardown
- Zum Vorbereiten und Aufräumen gemeinsamer Objekte und Rollback



Asserts

Setup

Teardown

(optional)

Demo

Probleme mit TDD in VBA/Access

- Visual Basic Classic
  - Nicht vorhandenes **Tooling** (Refactoring, CodeCoverage)
  - Keine **Reflection-API** → keine Isolationframeworks
- Access
  - Business-Logik oft in Formularen
  - Traditionell nur wenig Code ohne Datenzugriff
  - Häufig durchwachsene Codequalität
  - Geringes Bewusstsein für technisches Design (z.B. viele Responsibilites)

Unit Testing: Dos, Don'ts & Tipps

Was besser tun, lassen und überlegen

1/2

Was macht einen guten Unit Test aus?

Ein guter Unit Test ...

- ist leicht **verständlich**
- formuliert Bedingungen **deklarativ**
  - erhöhte **Lesbarkeit**, weniger fehleranfällig
- enthält **keine Logik**
  - Entscheidungsstrukturen → mehrere Tests
  - Schleifen → anderes/eigenes Assert verwenden
  - logische Ausdrücke → Is.LessThan, oä.
- testet genau eine Sache (**single Assert**)
- **trennt** Actions von Asserts
  - Arrange → Act → Assert (AAA)
  - Setup → Execute/Exercise → Verify (→ Teardown)

2/2

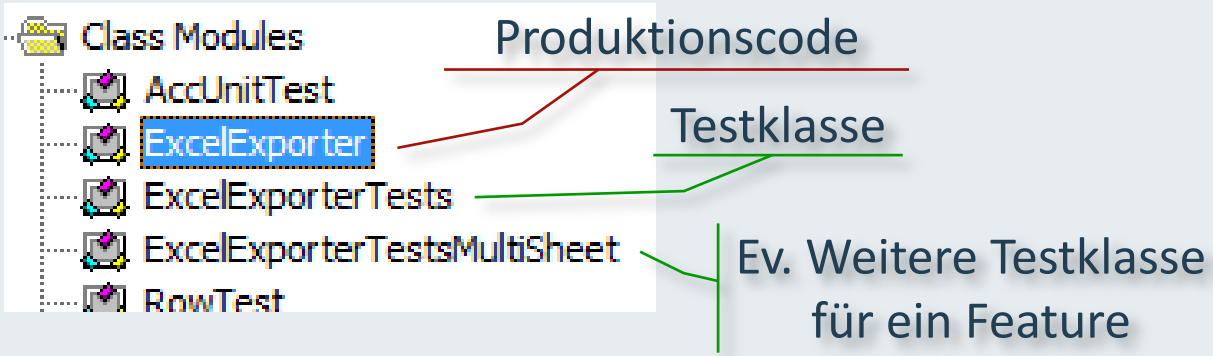
Was macht einen guten Unit Test aus?

Ein guter Unit Test ...

- hat eine **kurze Laufzeit**
- läuft komplett **im Hauptspeicher**
- läuft **automatisiert**
- kann **schnell ausgeführt** werden
- ist **unabhängig** von anderen Tests
  - unabhängig von der **Reihenfolge** der Testausführung
  - teilt sich **keinen State** (≠ Datenbank und Filesystem!)
- ist **vertrauenswürdig**
  - wenn er fehl schlägt, liegt ein Bug im Produktionscode vor
- macht **keine** über den Testcode hinausgehende **Annahmen**
  - keine Konfiguration nötig

Benennung und Organisation von Tests

- Test-Klassen (Fixtures)



- Test-Methoden

Getestete Methode | Getesteter Zustand | Erwartetes Verhalten

```
Public Sub GetRefund_WithNegativePercentage_ThrowsException()
    ' ...
End Sub
```

Test-First Development

Der Test soll fehlschlagen! – Vorerst zumindestens

Begriff: Refactoring

- Verändern von Code
  - **ohne** die **Funktionalität** zu **verändern**
  - in Hinblick auf **verbesserte Codestruktur**

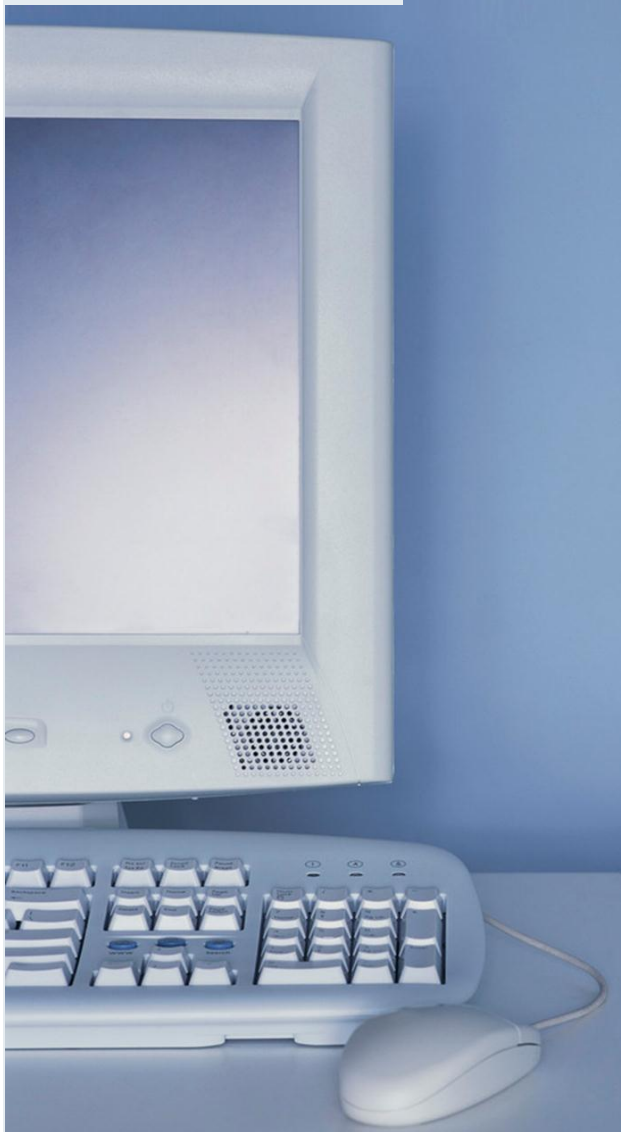
- Wichtige Refactorings
  - Rename
  - Move Method
  - Extract Method
  - Encapsulate Field
  - …

Test-First Development

- Der Test wird **vor** dem Produktionscode geschrieben
- **Red** → **Green** → **Refactor**:
  - Red: Der Test wird geschrieben und schlägt fehl (kein Produktionscode vorhanden)
  - Green: Der Produktionscode wird implementiert „Do the simplest thing that could possibly work."
  - Refactor: Der Produktionscode wird refactored (der Test bleibt grün!)

Ausnahme: **Refactoring**

Am Produktionscode darf nur dann etwas verändert werden, wenn es einen fehlschlagenden Unit Test gibt



Test-First Development

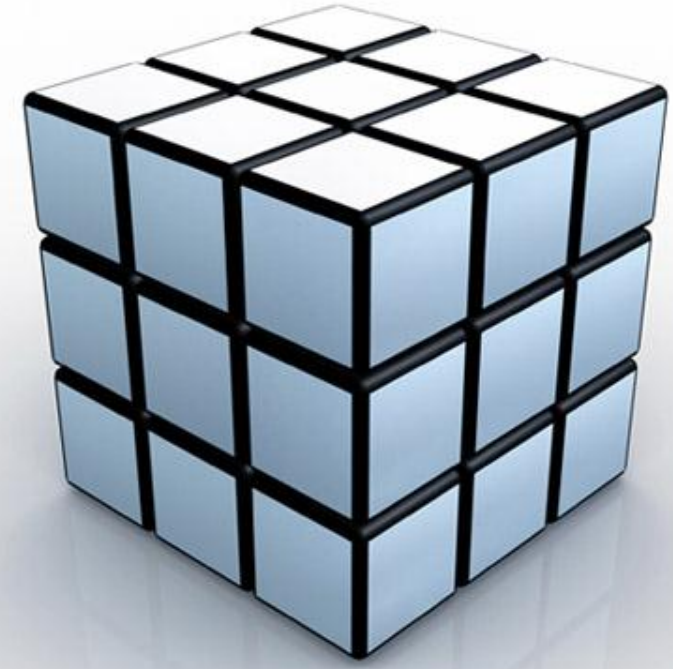
Aus rot wird grün
Aus grün wird ~~schün~~ schön!

Demo

IT WORKS – WE CARE
TECHTALK

Fragen?

IT WORKS – WE CARE
TECHTALK



... oder zuvor
beim **Abendessen**
und an der **Hotelbar**!
☺

Vielen Dank und – bis spätestens morgen!

Paul Rohorzka

pro@techtalk.at

IT WORKS – WE CARE
TECHTALK

Pragmatisches Testen von Access-Anwendungen

Teil 2

Agenda Teil 2

- Mit welchen Daten testen?
- Testen von Code mit Datenzugriff
- Testen in einer bestehenden Anwendung

Rekapituliere: Nutzen von Tests

- **Fokus** auf eine Einheit
  - Welche Szenarien sind denkbar?
  - Wie soll die Einheit reagieren? – Anforderungen!
- Größeres **Vertrauen** in den Code
- Größere **Sicherheit** bei Änderungen
- **Automatisierte** Regressionstests

- Tests als Dokumentation
- Tests zur Absicherung von Annahmen

Mit welchen Daten testen?

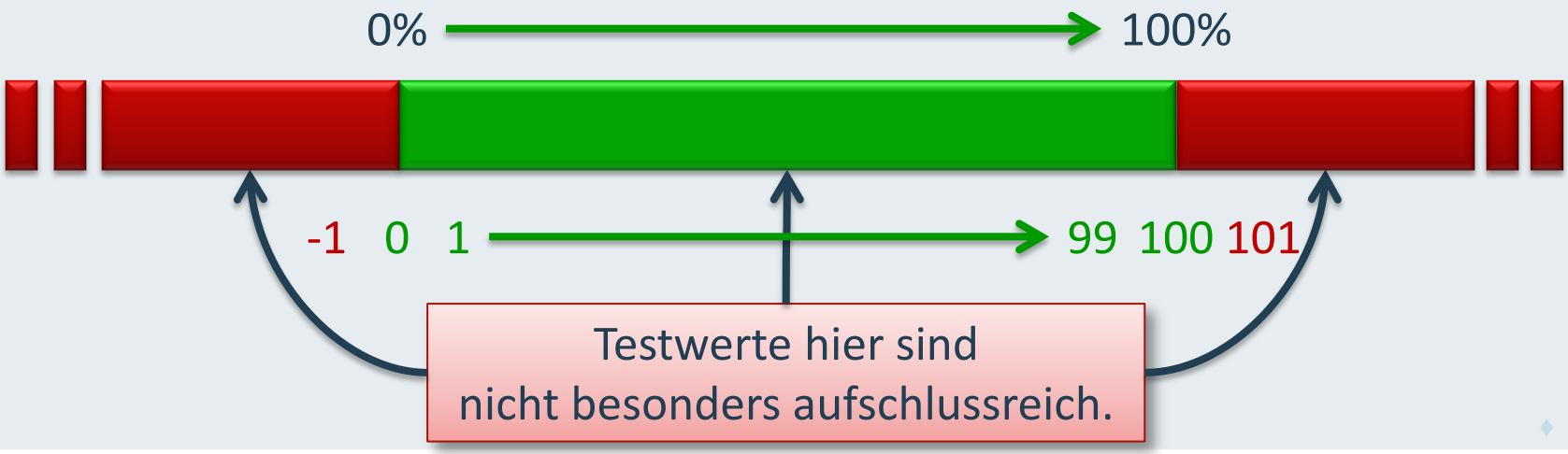
Woher sinnvolle Daten nehmen?

Mit welchen Daten testen?

- Beispiel:

```
Private Function GetDiscount _
    (ByVal Percentage As Integer) As Currency
     ' ...
End Function
```

- 2 Grenzwerte, 3 Äquivalenzklassen:



AccUnit: RowTest

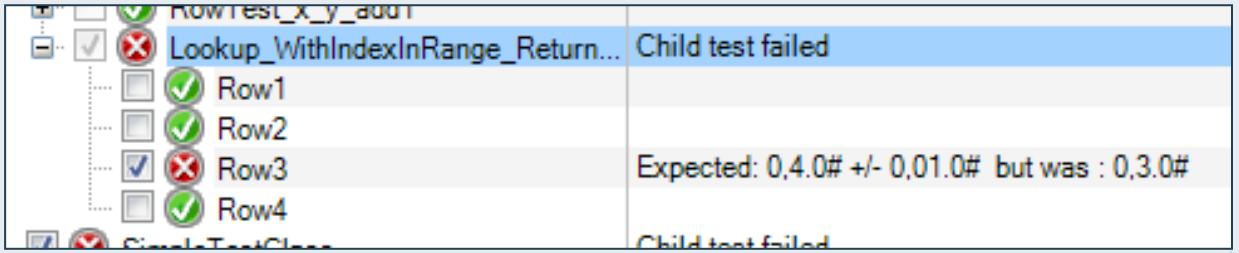
- Der selbe Test kann mit mehreren Daten ausgeführt werden:

```
'Lookup_WithIndexInRange_ReturnsItem(0, 0)
'Lookup_WithIndexInRange_ReturnsItem(1, 0.1)
'AccUnit:Row(2, 0.4)
'AccUnit:Row(3, 0.9)
Public Sub Lookup_WithIndexInRange_ReturnsItem _
    (ByVal Index As Integer, _
     ByVal ExpectedValue As Double)
  ' ...
End Sub
```

Variante mit **IntelliSense**

Unabhängige Variante (Umbenennen!)

- AccUnit:



Datengetriebene Tests

Ein Test, unterschiedliche Daten

Demo

Testen von Code mit Datenzugriff

Ran an die Datenbank!

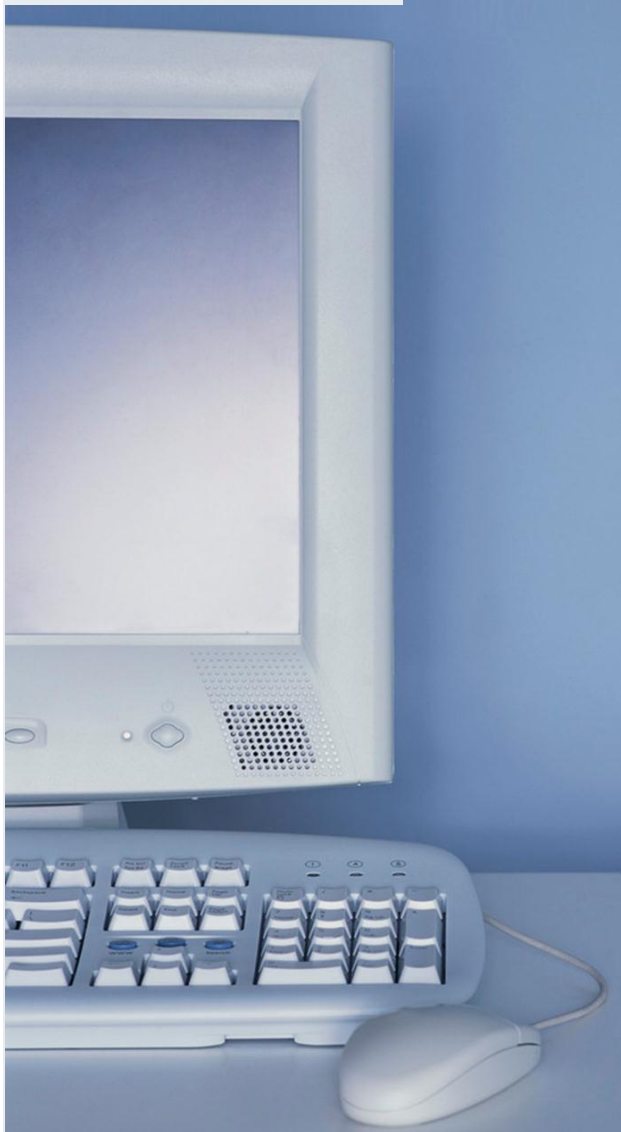
Was ist ein Integrationstest?

- Integrationstests
  - testen das Zusammenspiel mehrerer Einheiten
    - verschiedene Codeteile
    - Code und externes System (**Datenbank**!, Server, ...)
  - sind keine Unit Tests mehr!
  - brauchen oft eine passende Konfiguration

    → Fehlerquelle, nicht vertrauenswürdig
  - laufen meist langsamer
  - benötigen oft eine Komponente die sie nicht komplett kontrollieren können

Integrationstests mit Datenzugriff

- Wichtig: **Unabhängigkeit** der Tests!
- Lösungsansätze:
  - **Testdaten** nach jedem Test **explizit löschen** (aufwändig, fehleranfällig, schlecht zu warten)
  - **Backend wegwerfen** und durch **neues** ersetzen
    - einfach mit Jet
    - aufwändiger mit SQL-Server (detach, copy, attach)
  - **Transaktion** verwenden und **zurückrollen**
    - Nicht für Code der selbst Transaktionen verwendet
    - Vorsicht: DLookup() et al laufen außerhalb der Transaktion!
    - Vorsicht: AutoNumber-Seeds werden nicht zurückgesetzt!
    - AccUnit: Rollback-Attribut



Integrationstests mit Datenzugriff

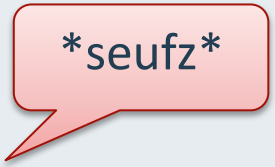
Demo

Testen in einer bestehenden Anwendung

Wo beginnen? – Und wie?

Wo mit dem Testen einer bestehenden Anwendung beginnen?

- Wie immer: 80/20
  - 80% der Bugs finden sich in 20% des Codes
  - Dort beginnen!
  - Codeteile mit hoher logischer Komplexität
- Bestehende Funktionalität absichern mit Integrationstests
- Neues Feature test-first entwickeln: Red → Green → Refactor



IT WORKS – WE CARE
TECHTALK

Fragen?

Akronyme

- Testing
  - TDD – **T**est **d**riven **d**evelopment
  - SUT – **S**ystem **u**nder **t**est
  - CUT – **C**ode **u**nder **t**est/**C**lass **u**nder **t**est
  - AAA – **A**rrange-**a**ct-**a**ssert
  - ATDD – **A**cceptance **t**est **d**riven **d**evelopment
  - BDD – **B**ehaviour **d**riven **d**evelopment
- Design
  - YAGNI – **Y**ou **a**in't **g**onna **n**eed **i**t
  - KISS – **K**eep **i**t **s**imple, **s**tupid
  - DRY – **D**on't **r**epeat **y**ourself
- Application Life Cycle Management
  - CR – **C**hange **r**equest
  - FR – **F**eature **r**equest

Literatur

- Bücher
  - Roy Osherove, **The Art of Unit Testing** with Examples in .NET, Manning, 978-1-933988-27-6
  - Michael C. Feathers, **Working Effectively with Legacy Code**, Prentice Hall, 978-0-13-117705-5
  - Robert C. Martin, **Clean Code**: A Handbook of Agile Software Craftmanship, Prentice Hall, 978-0-13-235088-4
  - Martin Fowler, **Refactoring**: Improving the Design of Existing Code, Addison-Wesley, 978-0-201-48567-7
- Web
  - www.ISerializable.com – Roy Osherove's Blog
  - www.hanselminutes.com – Wöchentlicher Podcast

IT WORKS – WE CARE
TECHTALK

Vielen Dank!

Paul Rohorzka

pro@techtalk.at

TechTalk Software Support Handelsgesellschaft m.b.H.
Leonard-Bernstein Straße 10
1220 Wien

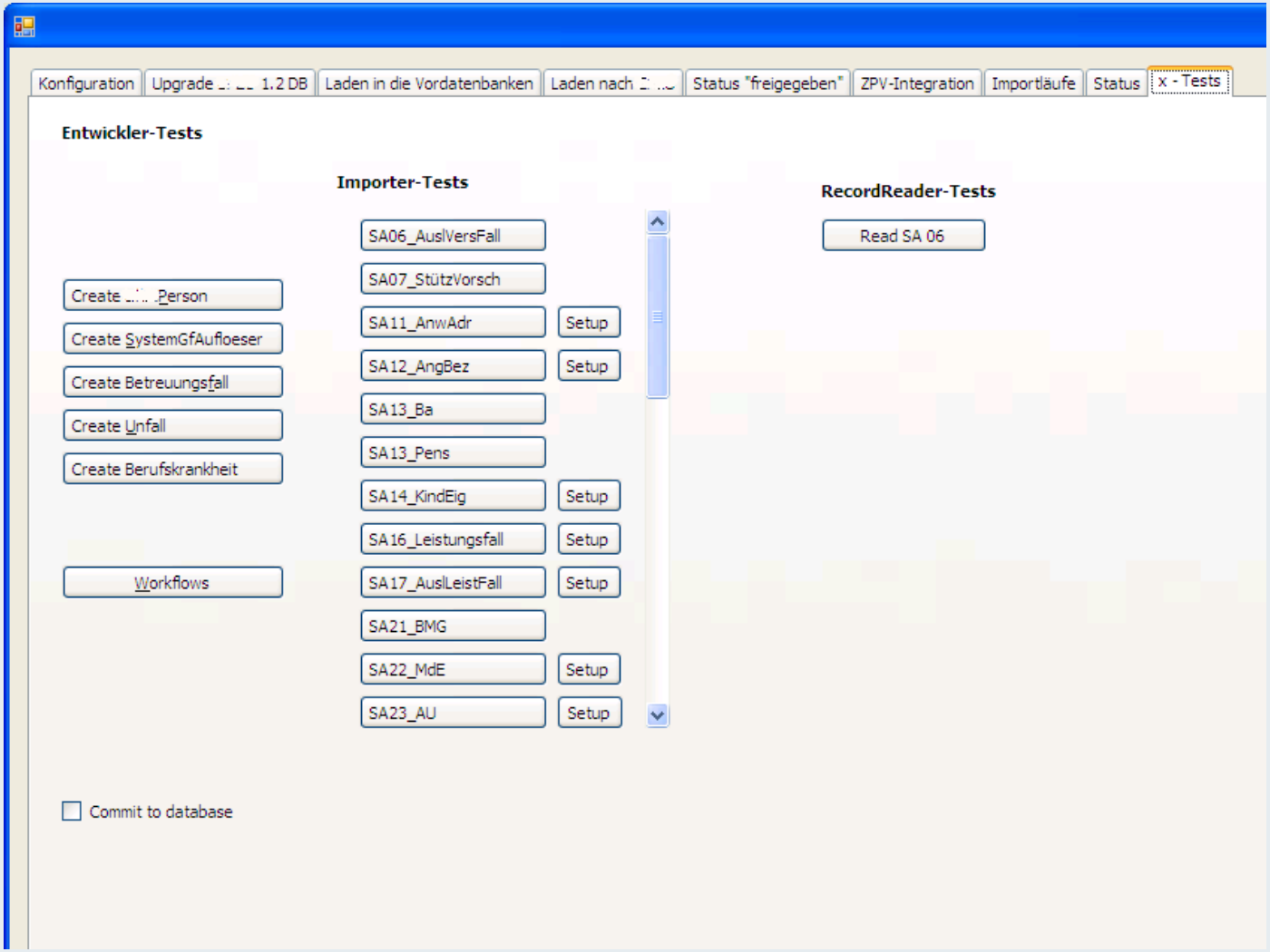
Was sich im Laufe von 3xAEK13 noch so angesammelt hat

Zusätzliches Material

- Code Archeologie
- Bild zur Motivation
- Covers der Buchtipps
  - Inkl. Hinweise auf deutschsprachige Ausgaben
- Follow-ups zu den Samstags-Sessions
- Ankündigung Workshop „Testen in Access“

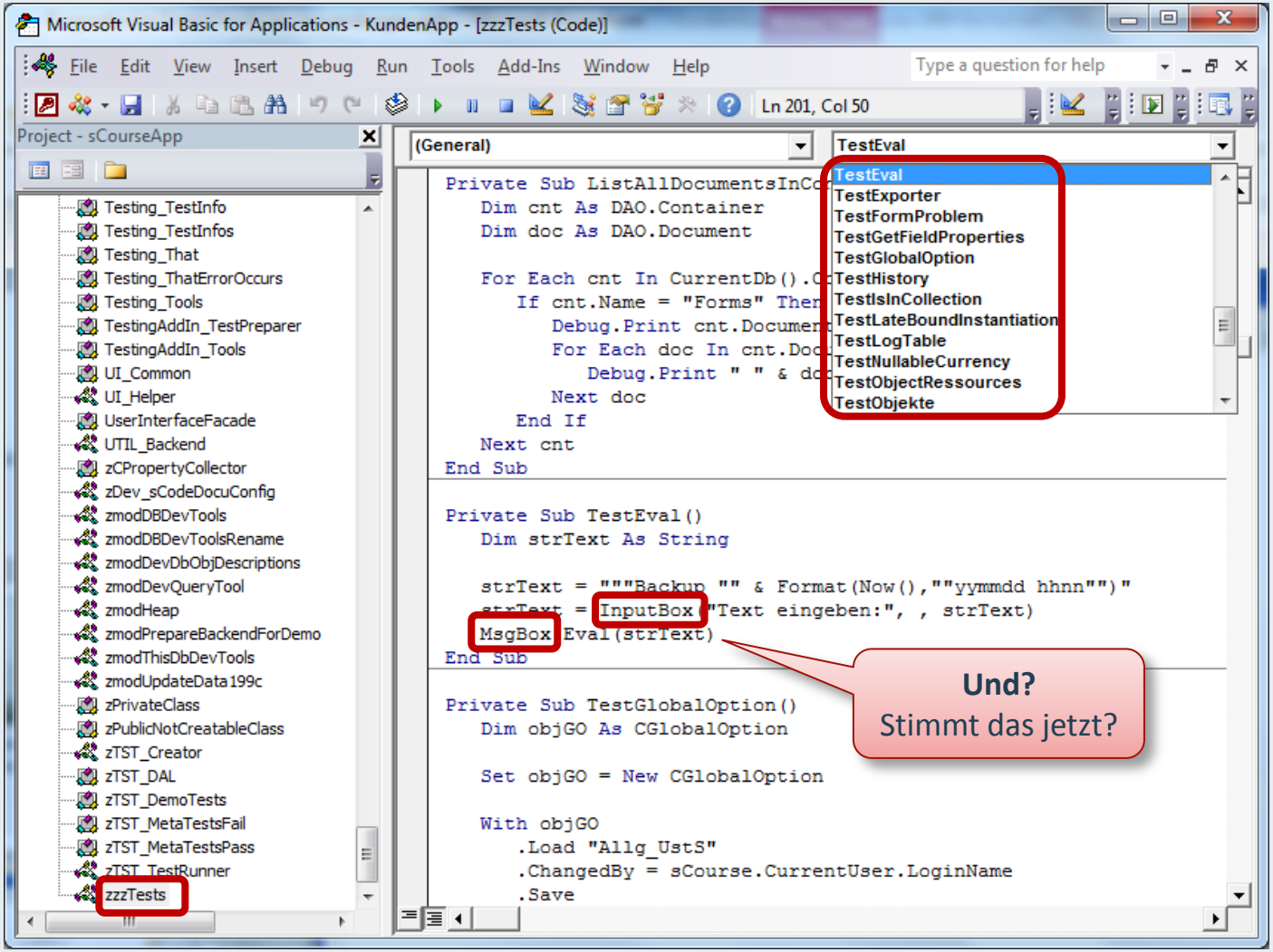
Code-Archäologie 1/2

Aus dem Fundus der Geschichte



Code-Archäologie 2/2

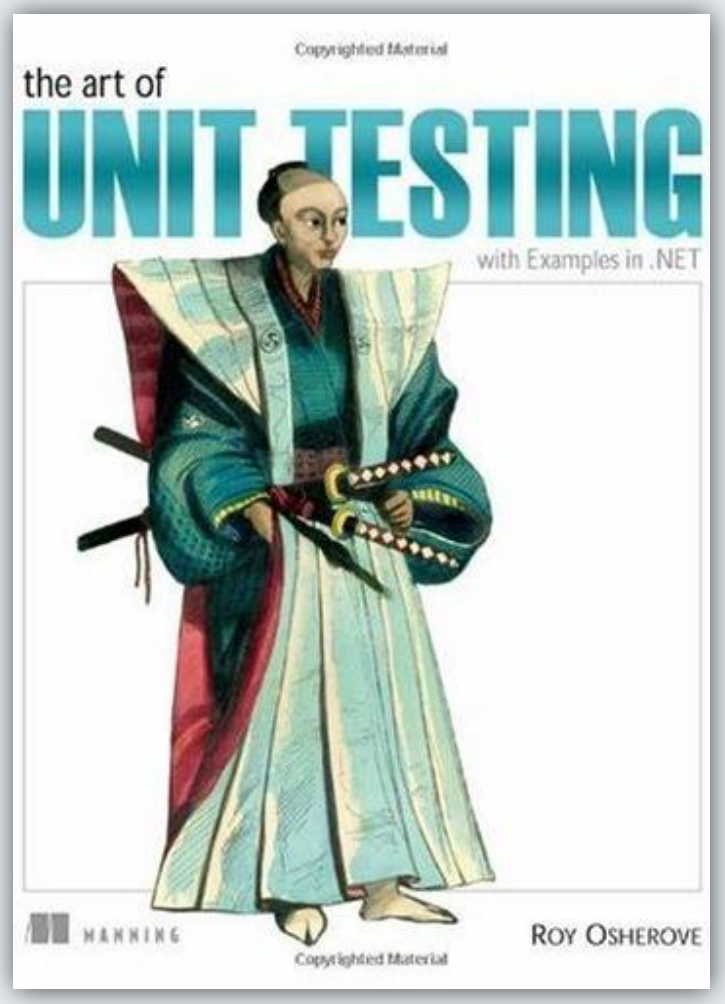
Aus dem Fundus der Geschichte



Ups, jetzt ist der Kunde aber ganz schön böhze

Das Bild zur Motivation

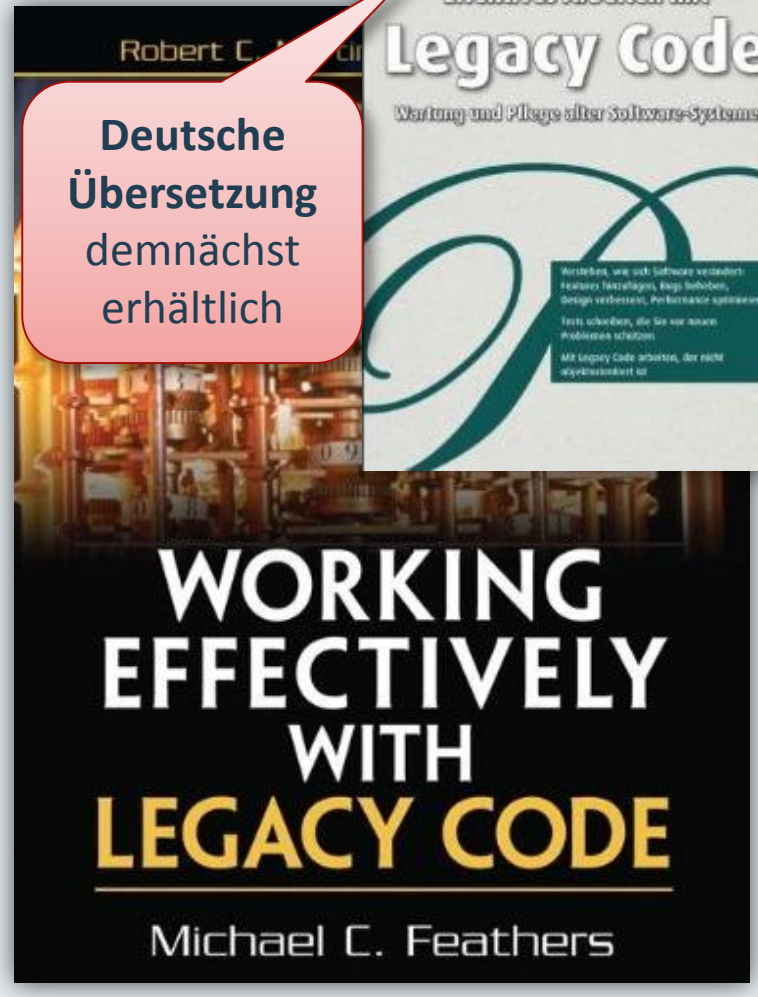
Covers der Buchtipps



2/2

Covers der Buchtipps

Deutsche Übersetzung vergriffen (?)



Deutsche Übersetzung demnächst erhältlich

AEK13A: Einschub

Follow-up zu gestern

- Aussprache "Teardown": \'ter-?dau?n\
- Ad Demo2:
  - Warum ist ein anderer Test fehlgeschlagen?
  - Wo ist der Fehler?
- Ad AccUnit
  - Verdoppelung der Codemodule durch Tests:
    - Feature: Export/Import Tests
    - Download SimplyVbUnit: Setup ist verfügbar!

AEK13C: Follow-up zu gestern

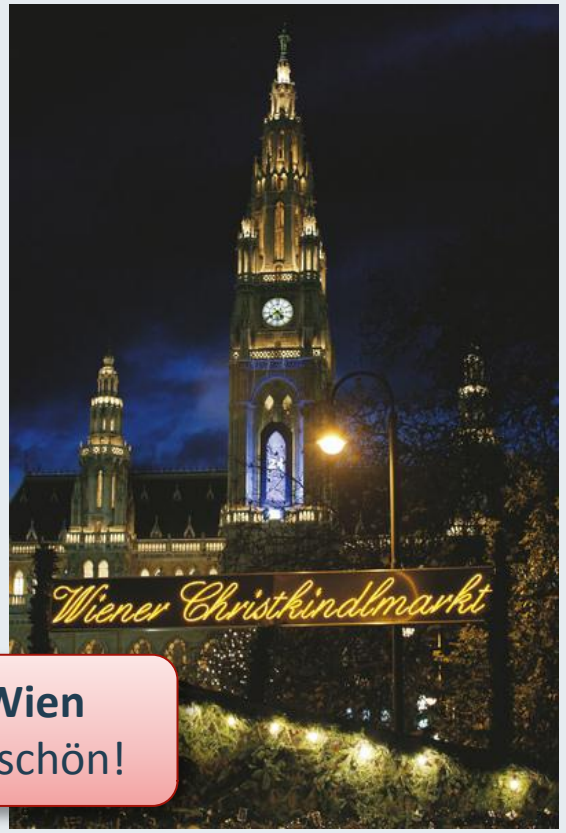
Ad Test First Development

- Testing in der Hotelbar

AEK13B&C – das Angebot gilt aber auch für AEK13A!

Workshop „Testen in Access“

- Inhalte
  - Anwendung testbar machen
  - Was konkret testen?
  - Wie testen?
  - Wie Änderungen implementieren?
  - Features von AccUnit
- Zwei Tage hands-on
- Bei Interesse:
  - Mail an pro@techtalk.at
    - Wunschzeitraum und –ort



Wien
ist schön!