AEK 11 - 11. Access Entwickler Konferenz
Nürnberg, 11./12., 18./19. und 25./26. Oktober 2008

Konfigurations- und Versionsmanagement in Microsoft® Access Projekten

Jenseits des Codes

Versionierung

Konfiguration

Dokumentation

softconcept, Wien
www.softconcept.at

Paul Rohorzka
paul.rohorzka@softconcept.at

Ziele

- Schärfen des Blicks für Softwareentwicklungs-Zyklen
  - Unterschiedliche Rollen
  - Verschiedene Bedürfnisse der Beteiligten
- Hilfestellungen für leichter wartbare Anwendungen
- Erhöhte Qualität im Betrieb der Anwendungen
- Kompetenteres Auftreten gegenüber IT-Abteilungen

Themen

- **Prolog:**
  Jenseits von Codd und Böse – unser Job von oben betrachtet
  - Phasen
  - Rollen
  - Herausforderungen
- **Teil I:**
  Versionierung
  - Warum versionieren?
  - Was versionieren?
  - Wie versionieren?
  - Versionierung von Backends

Themen (cont'd)

- **Teil II: Konfigurationsmanagement**
  - Bedürfnisse und Anforderungen
  - Wohin mit den Konfigurationsdaten?
  - Handhabung im Code
- **Teil III: Dokumentation**
  - Was dokumentieren?
  - Für wen dokumentieren?
  - Wie dokumentieren?

Prolog

Jenseits von Codd und Böse – unser Job von oben betrachtet

Phasen

Rollen

Herausforderungen

Wir entwickeln Software

„Erfolgreiche Software muss vor allem einmal tolle Features haben."

Oder...?

Gibt's da nicht vielleicht
auch noch ein Rundherum?

Phasen eines Softwareentwicklungs-Projekts

- Anforderungsanalyse
- Aufwandsschätzung
- Entwicklung
- Test

Rollout

Deployment

Konfiguration

Wartung

Für reale Projekte
selten nur
strikt aufeinander folgend
(Wasserfall)
...

stattdessen
einige mehrfach durchlaufene
Rückkopplungsschleifen
(Iterationen, Sprints, ...)

Abhängig vom gelebtem
Vorgehensmodell

=> Die Entwicklung ist (nur)
ein Bestandteil von mehreren
eines erfolgreichen Projekts!

Rollen in einem Softwareentwicklungs-Projekt (Auswahl)

- Auftraggeber
- Analyst
- Entwickler
- Designer
- Tester (fachlich, technisch)
- Rollout-, Deployment-, Configuration-Manager
- Support (1st-, 2nd-, 3rd-, ... level)
- Administrator
- Benutzer

=> Der Entwickler ist (nur) **eine Rolle von vielen** in einem erfolgreichen Projekt!

Je nach Projektgröße und -setup übernimmt eine Person mehrere Rollen.

Rollen:
Rollout-, Deployment- und Configuration-Management

- Rollout-Management
  - Releaseplanung
    - Was (regulär, Service Pack, Hotfix)?
    - Wann?
  - Durchführung
    - Durch wen?
  - Berücksichtigung von Abhängigkeiten
- Deployment-Management
  - Vorbereitung
    - Wie?
    - Abfolge?
- Administrator
  - Durchführung des Deployments
- Configuration-Management
  - Konfigurationen
    - vorbereiten
    - warten
    - dokumentieren

Rollout:
Arten von Releases

- Reguläre neue Versionen
- ServicePacks/Patches
  - Inhalt
    - Nicht kritische Bugs
    - Neue Features (Feature Request)
    - Veränderte Funktionalität (Change Request)
  - Gute Planung
- Hotfixes
  - Inhalt
    - Kritische Bugs
  - Parallel zur geplanten Weiterentwicklung
  - Zeitnahe Release
  - Gute Planung
    - Gefahr der „Verschlimmbesserung“

Rollen und Phasen: Testen

- Einige wichtige Arten von Tests
  - UnitTests
  - Integrationstests
  - Regressionstests
  - Smoketests
  - ...
- Wichtige Fragen
  - Wieviel testen?
  - Was testen?
  - Wer testet?
  - Wie testen?
  - Wann testen?

Praxisbeispiel: EFEU – Software aus dem Bereich der österreichischen Sozialversicherung

- EFEU – „**E**lektronische **F**eststellung und **E**rledigung in der **U**nfallversicherung"
- Entwickelt von der AUVA (Allgemeine Unfallversicherungsanstalt)
  - Für alle vier österreichischen Unfallversicherungs-Träger
- 6 Jahre Projektlaufzeit
- 20 Mio € Budget
- ~40 Personen Kernteam (Höchststand 56 Köpfe):
  - 7 Projektmanager + Assistenz
  - 1 Architekt
  - 2 Analysten
  - 13 Entwickler .NET
  - 3 Entwickler SAP
  - 6 Tester
  - 8 Fachbereichs-Mitarbeiter (Analyse, Test & 2nd-Level Support)
  - 1 3rd-Level Support
  - ? Infrastruktur

Praxisbeispiel: EFEU – Typischer Wartungszyklus

Wartungszyklus (anders für Planreleases):

1) Benutzer – *meldet Problem*
2) 1st & 2nd Level Support – *hilft oder leitet weiter*
3) 3rd Level Support – *hilft oder koordiniert Lösung*
4) Entwickler – *schätzt Aufwand*
5) Manager – *trifft Entscheidung über den Umfang*
6) Entwickler – *implementiert*
7) Build- und Configuration-Manager – *erstellt Setup für Q*
8) Administrator – *führt Deployment auf Q durch*
9) Tester – *testen Bugs nach*

Praxisbeispiel: EFEU – Typischer Wartungszyklus - cont'd

10) Fachadmin – *konfiguriert fachlich (ggf.)*

11) Fachliche Tester – *testen fachliche Richtigkeit*

12) Tester – *führen Regressions- und Integrationstests durch*

13) Rollout-Manager – *entscheidet über Rollout*

14) Build- und Configuration-Manager – *erstellt Setup für P*

15) Administrator – *führt Deployment auf P durch*

16) Tester – *führt Smoketests durch*

17) Fachadmin – *konfiguriert fachlich (ggf.)*

18) Benutzer – *benutzen die neue Release* ☺

Teil I

Versionierung

Warum versionieren?

Was versionieren?

Wie versionieren?

Versionierung von Backends

Warum versionieren?

- Vermeidung von Inkompatibilitäten
- Technische Überprüfbarkeit von Kompatibilität
  - Bei Programmstart (Backend!)
  - Jedenfalls: Vor der Verwendung eines Bestandteils
- Für jede Installation einer Software
  - Klarheit über den Zustand
  - Überprüfbarkeit des Status

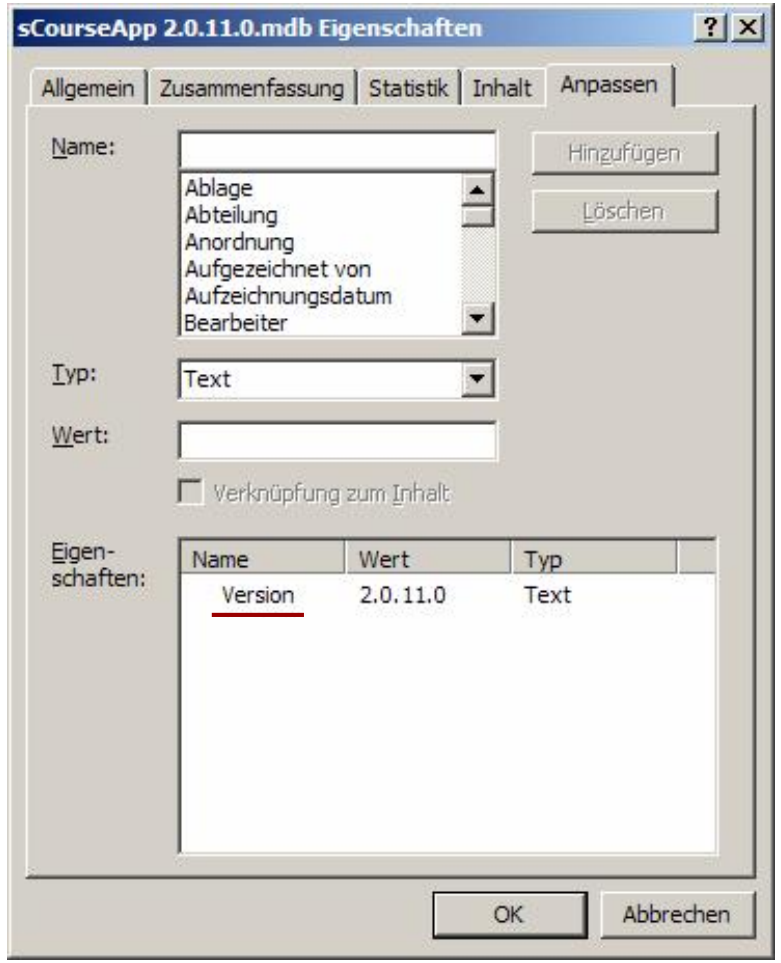
Was versionieren?

- Alle Hauptbestandteile
  - Frontend
  - Backend
  - Software-Komponenten
- Eventuell auch
  - Vorlagendokumente
  - Dateiformate (Konfigurationsdaten?)

$\Rightarrow$ Versionieren:
Alle eigenständigen Bestandteile der Software
(im weiteren Sinn)

Versionsnummer wo abspeichern?

- Möglichkeit für Access-Dateien: Benutzerdefinierte Datenbankeigenschaft



- Zugriff per Code:

```
CurrentDb()
  .Containers("Databases")
  .Documents("UserDefined")
  .Properties("Version")
```

Wann versionieren? („Die Goldene Regel der Versionierung“)

Alles, was aus der Hand gegeben wird, erhält eine eigene Versionsnummer.

- NIE:
  „Das ist eine neue 1.2er mit einer kleinen Änderung“
- Bewährte Vorgangsweise:
  Sofort nach der Weitergabe eine neue Versionsnummer vergeben
- KEINE Ausnahme:
  Weitergabe von Sourcen an andere Entwickler
  - (Andere Situation bei Verwendung einer Source Control)
- Alte Source-Stände einfrieren!
  - Nachstellen von Fehlern
  - Erstellen von Hotfixes

Vergabe von Versionsnummern

- Festlegen des Formats
  - z.B. Major.Minor (2.1)
  - z.B. Major.Minor.Build.Revision (2.1.467.3)
- Festlegen der Bedeutung
- Kompatibilität definieren
  - z.B. Major und Minor müssen übereinstimmen
  - z.B. Major muss übereinstimmen, Minor gleich oder höher
- Eventuell eigene Versionsbezeichnungen zu Marketingzwecken
  - z.B. Microsoft PowerPoint
    - 2002 SP3
    - entspricht Version 10.6842.6845

Versionsmanagement: Dokumentieren!

- Bestandteile der Software (Frontend, Backend, ...)
  - Change-Log (inkl. interner Änderungen)
  - History/Release Notes (für Endbenutzer)
  - Update-Skripts (Backend)
  - Source Control
- Ausgelieferte Releases
  - Dokumentation aller Versionsstände
    - Für jede Umgebung
    - Aktuell halten
- Es muss jederzeit klar sein, welche Installation auf welchem Stand ist
  - Nachstellen von Fehlern!

Versionierung von Backends

- Genaueste Dokumentation!
- Versionsnummer speichern in
  - Datenbank-Eigenschaft
  - Versionstabelle (am besten gleich inkl. Historie)
- Updateskripts für automatisches Upgrade
  - Händisch mitführen
    oder
  - Auf Basis mit Hilfe eines Tools
    (z.B. MDB Struktur Manager, RedGate SQL Compare, ...)
    - Händisches Überarbeiten oft notwendig (Bsp: Änderung Feldname)
    - Immer kontrollieren!
  - Testen!
- Validierung
  - Automatisch beim Verbinden des Frontends
- Upgrade
  - Ev. (halb-) automatisches Hochheben eines alten Backends
  - Concurrency-Probleme bedenken (Locking)

Hilfsmittel (nicht nur) zum Versionsmanagement: softconcept scProject

- Versionsmanagement
- ToDos
- Release Notes ("History")
- Backendversionen und -änderungen
- Dokumentation von Deliveries

- → Bei Interesse: Mail an mich!

Teil II

Konfiguration

Bedürfnisse und Anforderungen

Wohin mit den Konfigurationsdaten?

Handhabung im Code

Warum konfigurieren?

- Abwandlung des Verhaltens einer Software
  - Ohne Änderungen an den Sourcen
  - Für spezielle Situationen vor Ort
- Betrieb der Software in unterschiedlichen Umgebungen
  - Technische Administration
    - Pfade, Server, Credentials, ...
- Der Kunde soll selbst Anpassungen vornehmen können
  - Fachliche Administration
    - Vorgaben
    - Regeln
  - Benutzer
    - Verhalten
    - Einstellungen der Benutzeroberfläche
    - Erscheinungsbild

Konfiguration: Einige mögliche Anforderungen

- Zentrale Wartbarkeit
  - Vorzugsweise unabhängig von der Hauptapplikation
- Wiederherstellung von Konfigurationseinstellungen
  - Unabhängig von den Nutzdaten
- Übernehmen von Konfigurationen
  - Zwischen Umgebungen
    - z.B. Entwicklung → Test → Integration → Produktion
    - Nachvollziehen von Fehlern (Support)!
  - Zwischen Benutzern
    - Überschreiben von korrupten/unsinnigen Benutzereinstellungen
      - Mit sinnvollen Standard-Werten
      - Mit alten Werten aus Backup
    - Verteilen von neuen Konfigurationen aus Test-Benutzer
    - Vorgaben für neue Benutzer

Unterschiedliche Bedürfnisse: Endanwender

- Was konfigurieren?
  - Farben
  - Fensterpositionen
  - UI-Einstellungen im Allgemeinen
  - Voreinstellungen
- Wie konfigurieren?
  - Implizit
    - Fensterpositionen
    - Letzte Auswahl einer Optionsgruppe
    - Zuletzt gewählter Datensatz, ...
  - Explizit
    - Farben
    - Profimodus, ...
- Wo konfigurieren?
  - Integriert im UI der Hauptapplikation
    - Für explizite Parameter
    - Ev. auch für implizite Parameter

Unterschiedliche Bedürfnisse: Fachlicher Administrator

- Was konfigurieren?
  - Fachliche Parameter
    - Prozentsätze
    - Stichdaten
    - ...
  - Eventuell auch Benutzeradministration
    - Anlegen von Benutzern
    - Rechteverwaltung
    - Zurücksetzen von Benutzereinstellungen
- Wo konfigurieren?
  - Integriert im UI der Hauptapplikation
  - Eigenes Frontend für Fachadmin

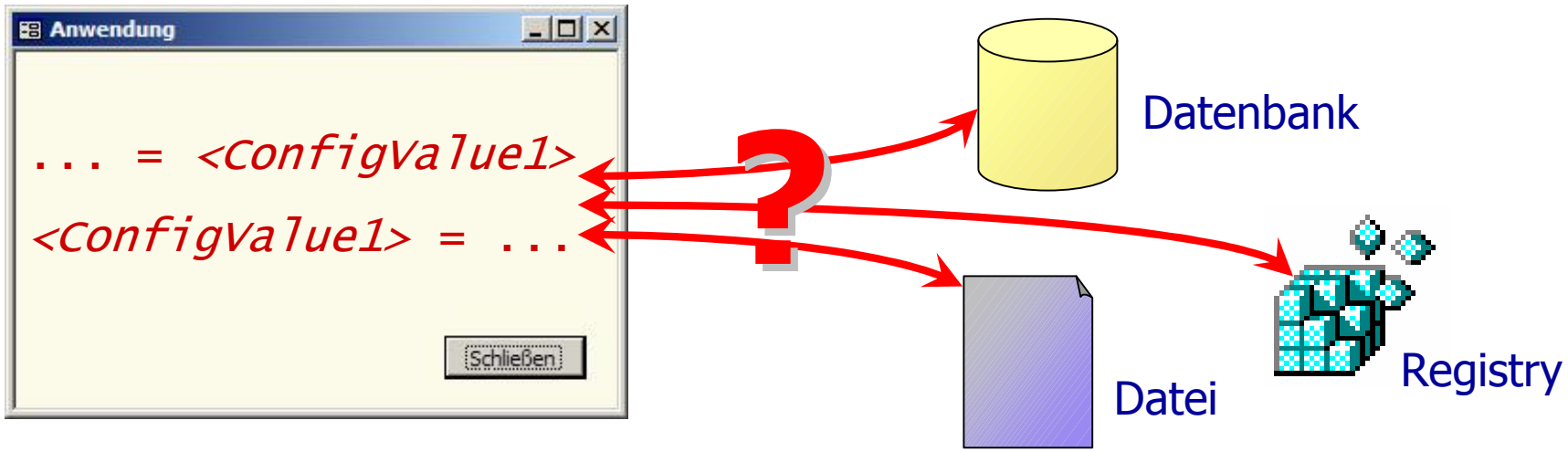
Konfiguration: Technischer Administrator

- aka „Betriebsführung“, ...
- Was konfigurieren?
  - Verzeichnisse
  - Drucker
  - Server
  - Logging
  - Timeouts
- Wo & wie konfigurieren?
  - Kein spezialisiertes UI (Editieren von Konfig-Dateien)
  - Eigenes Frontend (Admin-Client)
  - Eigenes UI für technischen Administrator im Applikations-Frontend?
    - Eigener Benutzeraccount in der eigentlichen Applikation notwendig!
    - Eher ungünstig
    - Problem: Völlig totkonfigurierte Anwendung ☹

Konfigurationsdaten: Ort der Speicherung

- Benutzerprofil (für Benutzerkonfiguration)
  - Registry (HKCU)
  - Dateien (%AppData%)
- Lokale ini- od. config-Datei
  - Schwierigkeiten und Probleme:
    - Eventuell auch für einfache Benutzer änder- und löschbar
    - Wartung (Verteilung)
    - Backup
- ini- od. config-Datei am Server
- Haupt-Backend
  - Problem:
    - Vermischung mit Daten
      → keine getrennte Wiederherstellung
- Eigenes Konfigurationsbackend

Konfiguration: Wie persistieren?



- Trennen:
  - Funktionalität Werte zu lesen und zu setzen
  - Frage wo die Daten wie gelesen und gespeichert werden
- Details der Speicherung
  - Art (Datenbank, Datei, Registry)
  - Ort (Server, Pfade, Keys)
  - Caching ja/nein
  - ...

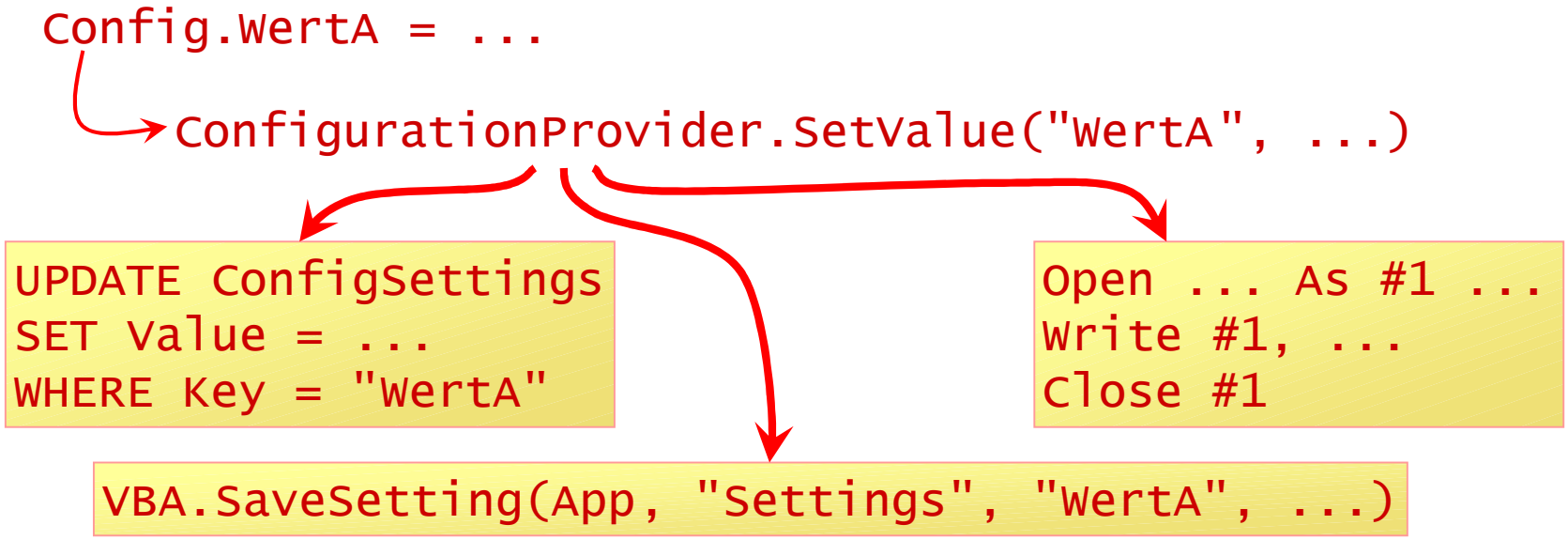
Konfiguration: Wie persistieren? – cont'd

- Realisierung der Trennung
  - Konfigurations-Code-Modul austauschen
    - modConfigRegistry, modConfigDatabase, modConfigFile, ...
    - Nachteil: Eingriff in den Code notwendig
  - Art der Speicherung im Code abfragen

```
Sub WriteSetting(Key, Value)
   Select Case ConfigMode
      Case ConfigDatabase
         WriteSettingToDatabase(Key, Value)
      Case ConfigRegistry
         WriteSettingToRegistry(Key, Value)
      ...
   End If
End Sub
```

Konfiguration: Wie persistieren? – cont'd

- Realisierung der Trennung – cont'd
  - Dependency Injection
    - ConfigurationProvider
    - Einfacher Austausch der Art der Speicherung
    - Per Konfiguration möglich ☺
    - Mit etwas Aufwand auch im Nachhinein



Konfiguration: Vermeidung von literalen Schlüsseln

- Keine literalen Schlüsseln direkt vor Ort

```
Me.Section(acDetail).BackColor = Config.BackColor
```

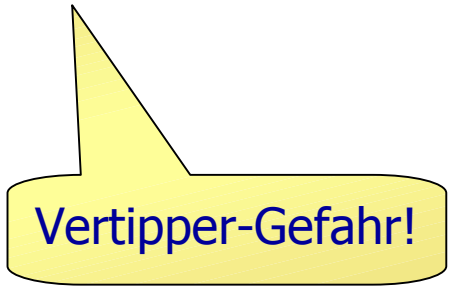
statt

~~`Me.Section(acDetail).BackColor = ...("BackColor")`~~

- Keine Gefahr von Vertippern
  - Ggflls. Compilerfehler
- Spezialisierte Konfigmodule
  - Benannte Properties für Konfiguration

```
Public Property Get BackColor As Long
   BackColor = ConfigurationProvider.GetValue("BackColor")
End Property
Public Property Let BackColor(NewBackColor As Long)
   ConfigurationProvider.SetValue("BackColor", NewBackColor)
End Property
```

- Ev. von Tool generieren lassen



Teil III

Dokumentation

Was dokumentieren?

Für wen dokumentieren?

Wie dokumentieren?

Dokumentation: Betriebshandbuch

- Für Betriebsführung (technische Administration)
- Obligatorisch
- Inhalt
  - Betroffene Version
  - Technische Voraussetzungen
    - Hardware-Voraussetzungen
    - Version Betriebssysteme (Client und Server)
    - Netzwerk (Novell!?)
  - Betriebsparameter
    - Zu erwartende Last und Datenmengen
    - Betriebszeiten
    - Wartungsintervalle
  - Jet-Backend: Stabilität der Infrastruktur (!)

Dokumentation: Betriebshandbuch (cont'd)

- Inhalt – cont'd
  - Versionen benötigter Software
    - Office (Service Packs!)
    - Komponenten Jet, MDAC, ActiveX, ...
  - Organisatorische Voraussetzungen
    - Ansprechpartner
      Verfügbarkeit
      Kompetenzen
- Form
  - PDF
  - Kunden-Vorgabe (XML)
- Nebenwirkung
  - Hilfsmittel zur Argumentation bei Unstimmigkeiten (für beide Seiten!)

Dokumentation: Fachliche Administration und Benutzerdokumentation

- Dokumentation zur fachlichen Administration
  - Obligatorisch
    - Begründung: Auswirkungen auf fachliche Richtigkeit von Berechnungsergebnissen
  - Ggffls. als Teil der Benutzerdokumentation
- Endbenutzer (Benutzerhandbuch/Onlinehilfe)
  - Muss explizit vereinbart werden
  - Hoher Aufwand
  - Fragwürdiger Nutzen (*„Doku? Ach da schau ich nie hinein!"*)
  - Alternativen:
    - Schulung
    - Integriertes System zur Selbstdokumentation
    - Wiki
    - Blog
    - Newsletter

Fragen, Kommentare?

Danke für die Aufmerksamkeit!