# TAV 43

# Modellbasierte Testdatenspezifikation und -generierung mittels Äquivalenzklassen und SQL

Dierk Ehmke und Mario Friske

# Agenda

- Fallbeispiel
- Problemstellung / bekannte Ansätze
- Unsere Ansätze
  - Spezifikation mittels Äquivalenzklassen
  - Generierung mittels SQL
- Zusammenfassung & Ausblick

# Ausgangslage

- Schwammige Anforderungen
- Agil: unklare Testbasis: unaktuelle Anforderungen
- Komplexe Fachdomäne
- Testeinschränkungen für Testdaten wie
  - vorgegebene Testdatensätze (z. B. Testkunden) und
  - Mockregeln (IBAN endet mit 0 für schlechte Bonität)
- Aufwand für bzgl. Testidee unwesentliche Details
- Hoher Wartungsaufwand

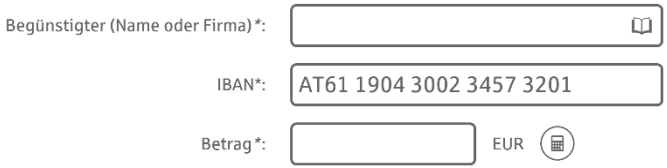
# Ziel

- Fachlich realistische Testdaten
- Knackige, genaue Beschreibung der Fachlichkeit, mit der Testdaten erzeugt werden können
- Möglichkeit, Testeinschränkungen zu berücksichtigen
- Niedriger Wartungsaufwand

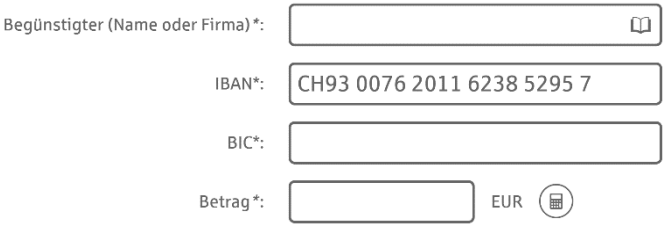
# Fallbeispiel IBAN-Überweisung: Länderspezifischer Masken-Workflow


**Leere Eingabemaske**


**Deutsche IBAN**




**Östereichische IBAN**




**Schweizer IBAN**

# Bekannte Ansätze zur Testdatenerzeugung

- Zufallsbasierte Erzeugung: sehr einfach
- Manuelle Erstellung: u.E. sehr häufig angewendet
- Toolunterstütze interaktive Erzeugung, d.h. Hilfe bei Selektion und Definition
- Regelbasierte Generierung, d.h. konstruktive Erzeugung großer Datenmengen
- Evolutionäre Methoden, d.h. systematisches Suchen und Rekombinieren
- Constraintsolving, d.h. Lösung eines Constraint satisfaction problem

# Unsere Ansätze zur Testdatenerzeugung

Kombination von

1. Interaktive Spezifikation auf Basis von Äquivalenzklassen und Attributierten Klassifikationsbäumen
   - Modellbasierte Beschreibung der Äquivalenzklassen
   - Partielle Testdatenspezifikation,
   - Vorgehen gemäß UCV-Methode
2. Regelbasierte Generierung auf Basis von SQL
   - Auffüllen der partiellen Testdatenbeschreibungen mittels Constraint Solving
   - Umsetzung im Werkzeug periplus.autogen

# Aufbau IBANs in DACH

| | | |
|---|---|---|
| Austria | AT61 1904 3002 3457 3201 | |
| | ATkk bbbb bccc cccc cccc | (16n bzw. 5n, 11n) |
| Germany | DE89 3704 0044 0532 0130 00 | |
| | DEkk bbbb bbbb cccc cccc cc | (18n bzw. 8n,10n) |
| Switzerland | CH93 0076 2011 6238 5295 7 | |
| | CH78 0055 40A1 0245 0260 1 | |
| | CHkk bbbb bccc cccc cccc c | (5n,12c) |

b = National bank code n = numeric characters

c = Account number c = mixed-case alpha-numeric characters

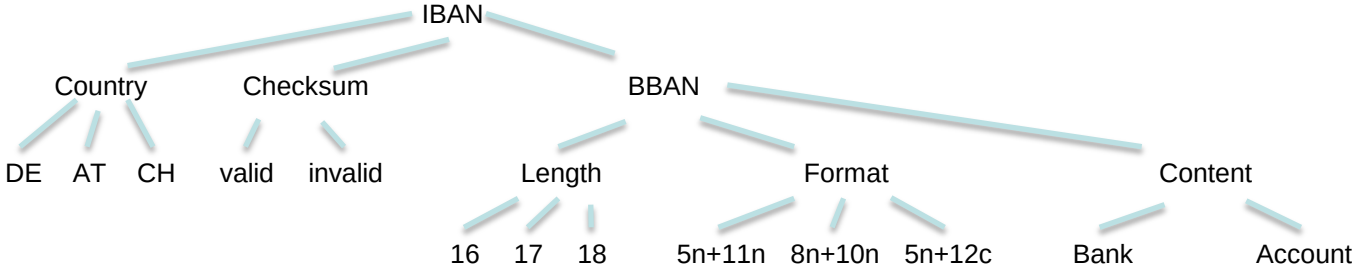
Quellen

https://en.wikipedia.org/wiki/International_Bank_Account_Number

http://www.swissiban.com/de.htm

https://www.xe.com/ibancalculator/

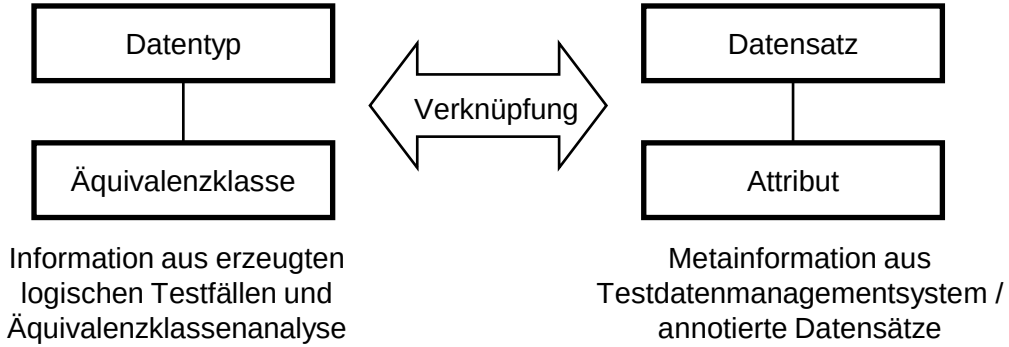
# Attributierte Äquivalenzklassen





siehe auch [Lützkendorf, Bothe: Attributierte Klassifikationsbäume zur Testdatenbestimmung. 2003]

# Interaktive Verknüpfung



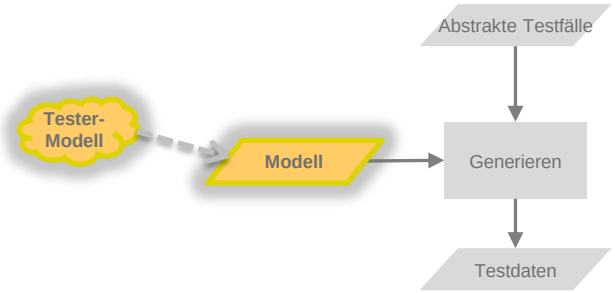

- Interaktive Erstellung und Verknüpfung mit Szenarien (Pfaden durch Use Case)
- Problematisch sind partiell spezifizierte Testdaten => Verfahren zur Vervollständigung erforderlich

# SQL als Beschreibungssprache für Testdatenspezifikationen

- Entworfen für Beschreibung komplexer Datenflüsse
- Große Verbreitung unter Entwicklern und (technischen) Testern
- Kompakte textuelle Darstellung (Bildschirm, Drucken, Versionierung)
- Deklarative Spezifikation, gutes und einheitliches Abstraktionsniveau


www.integrant.com/sql-20052008-master-database-recovery

# Schema MBTD


Abstrakte Testfälle
Tester-
Modell
Modell
Generieren
Testdaten

# Beispiel IBAN

- Ziel: Generierung von IBAN’s, die im Test akzeptiert werden
- Formulieren der Zusammenhänge (Constraints und INSERT-Statements)
- Ausgabeformatierung festlegen
- Festlegen der Testfälle (nur testrelevante Spalten)


Mario Friske
Freelancer
Berlin



Dierk Ehmke
periplus instruments

# Modell IBAN

```
CREATE TABLE ACCOU(
  IBAN          CHAR(34),
  BANK_CODE     DECIMAL(20,0),
  ACCOUN        DECIMAL(20,0),
  COUNTRY       CHAR(100),
  CONSTRAINT check_iban
    CHECK( COUNTRY IN('GERMANY','AUSTRIA','SWITZERLAND') )
);

CREATE TABLE ACCOU2(
  IBAN          CHAR(34),
  BANK_CODE     DECIMAL(20,0),
  ACCOUN        DECIMAL(20,0),
  COUNTRY       CHAR(100),
  CHECKNO       DECIMAL(5,0),
  COUNTRYMARK   CHAR(4),
  COUNTRY_NO    DECIMAL(6,0),
  ACC_FACTOR    DECIMAL(20,0),
  BC_FACTOR     DECIMAL(20,0),
  NP_FORMAT     CHAR(34),
  NUMERIC_PART CHAR(34) not null,
  CONSTRAINT check_iban2
  CHECK(
        ( BANK_CODE >= 1 )
    AND ( ACCOUN >= 1 )
    AND ( CHECKNO >= 0 )
    AND ( COUNTRYMARK IN('AT','CH','DE') )
    AND ( COUNTRY_NO IN(1314,1029,1217) )
    AND ( ACC_FACTOR IN(10000000000,100000000000,1000000000000) )
    AND ( BC_FACTOR IN(100000000,100000) )
    AND ( NP_FORMAT IN('00000000000000000000',
                       '00000000000000000000',
                       '00000000000000000000')
        )
```

```
-- http://www.pruefziffernberechnung.de/I/IBAN.shtml
-- http://ibanvalidieren.de/beispiele.html
```

```
    AND ( BANK_CODE <= BC_FACTOR - 1 )
    AND ( ACCOUN <= ACC_FACTOR - 1 )
    AND ( CHECKNO = 98 - mod ( ( BANK_CODE * ACC_FACTOR * 1000000 +
          ACCOUN * 1000000 + COUNTRY_NO * 100 ), 97 ) )
    AND ( CHECKNO * BC_FACTOR * ACC_FACTOR + BANK_CODE * ACC_FACTOR +
          ACCOUN = TO_NUMBER( NUMERIC_PART, NP_FORMAT ) )
    AND ( IBAN = CONCAT( COUNTRYMARK, NUMERIC_PART ) )
    AND (
         (      ( COUNTRYMARK = 'DE' )
            AND ( COUNTRY = 'GERMANY' )
            AND ( COUNTRY_NO = 1314 )
            AND ( ACC_FACTOR = 10000000000 )
            AND ( BC_FACTOR = 100000000 )
            AND ( NP_FORMAT = '00000000000000000000' )
         )
        XOR
         (
                ( COUNTRYMARK = 'AT' )
            AND ( COUNTRY = 'AUSTRIA' )
            AND ( COUNTRY_NO = 1029 )
            AND ( ACC_FACTOR = 100000000000 )
            AND ( BC_FACTOR = 100000 )
            AND ( NP_FORMAT = '00000000000000000000' )
         )
        XOR
         (
           ...
         )
        )
    )
);

INSERT INTO ACCOU2 SELECT IBAN, BANK_CODE, ACCOUN, COUNTRY FROM ACCOU A;
```

```
-- https://www.iban-bic.com/sample_accounts.html
-- https://de.wikipedia.org/wiki/Internationale_Bankkontonummer
```

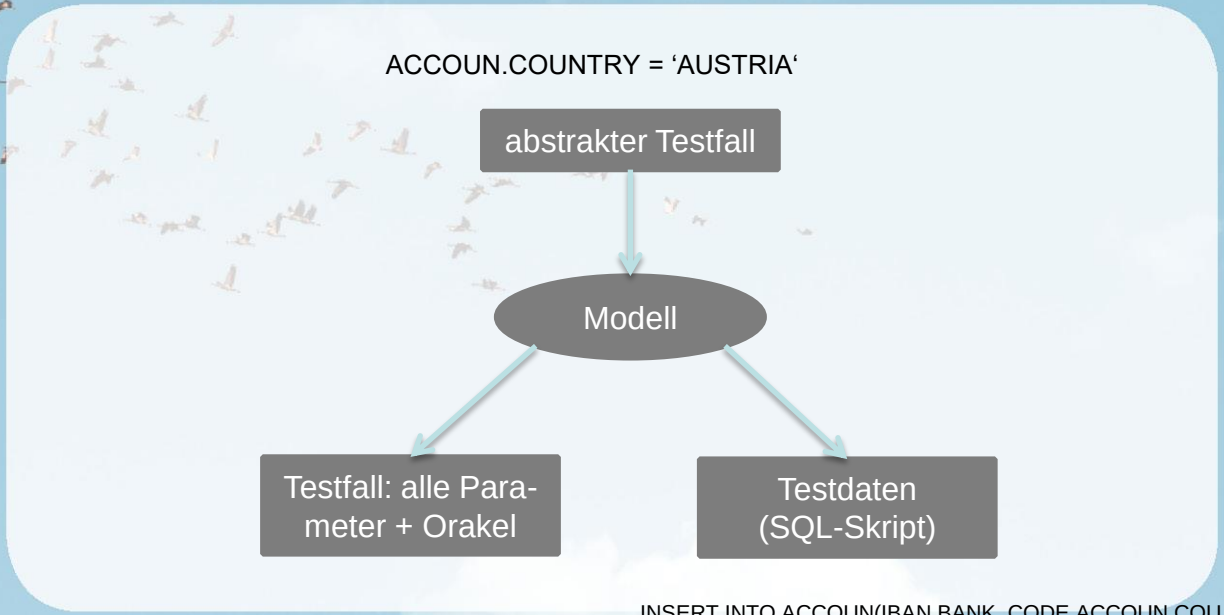
# Partielle Testdatendefinition & Generierung

| No | Comment | BANK_CODE | ACCOUN | COUNTRY | IBAN |
|---|---|---|---|---|---|
| 1 | BANK_CODE, ACCOUN and COUNTRY preset | 64090 | 12345678901 | AUSTRIA | AT896409012345678901 |
| 2 | only BANK_CODE preset, fits only to DE | 64090100 | 1 | GERMANY | DE886409010000000001 |
| 3 | only ACCOUN preset, fits to DE | 1 | 1234567890 | GERMANY | DE170000000011234567890 |
| 4 | only ACCOUN preset, fits to CH | 1 | 123456789012 | SWITZERLAND | CH4300001123456789012 |
| 5 | only COUNTRY preset | 1 | 1 | SWITZERLAND | CH1700001000000000001 |
| 6 | only IBAN preset | 20050550 | 1015871393 | GERMANY | DE02200505501015871393 |
| 7 | smoke test, nothing preset | 1 | 1 | SWITZERLAND | CH1700001000000000001 |

**Minimaler Aufwand für Wartung**

preset generated

# Vom abstrakten zum konkreten Testfall: MTDG ↔ MBT




```
INSERT INTO ACCOUN(IBAN,BANK_CODE,ACCOUN,COU
NTRY)  VALUES (‘AT500000100000000001’;1;1;’AUSTRIA’);

INSERT INTO CUSTOM(NO,IBAN,…)  VALUES ('000000000
001','1234567890',…);
```

periplusautogen

# Testfallgenerierung

| No | Comment | BANK_CODE | ACCOUN | COUNTRY | IBAN |
|---|---|---|---|---|---|
| 1 | pairwise test step | 1 | 1 | SWITZERLAND | CH1700001000000000001 |
| 2 | pairwise test step | 1 | 1 | GERMANY | DE440000000100000000001 |
| 3 | pairwise test step | 1 | 1 | AUSTRIA | AT500000100000000001 |

**Minimaler Aufwand für Testsystematik/Kombinatorik**

preset generated

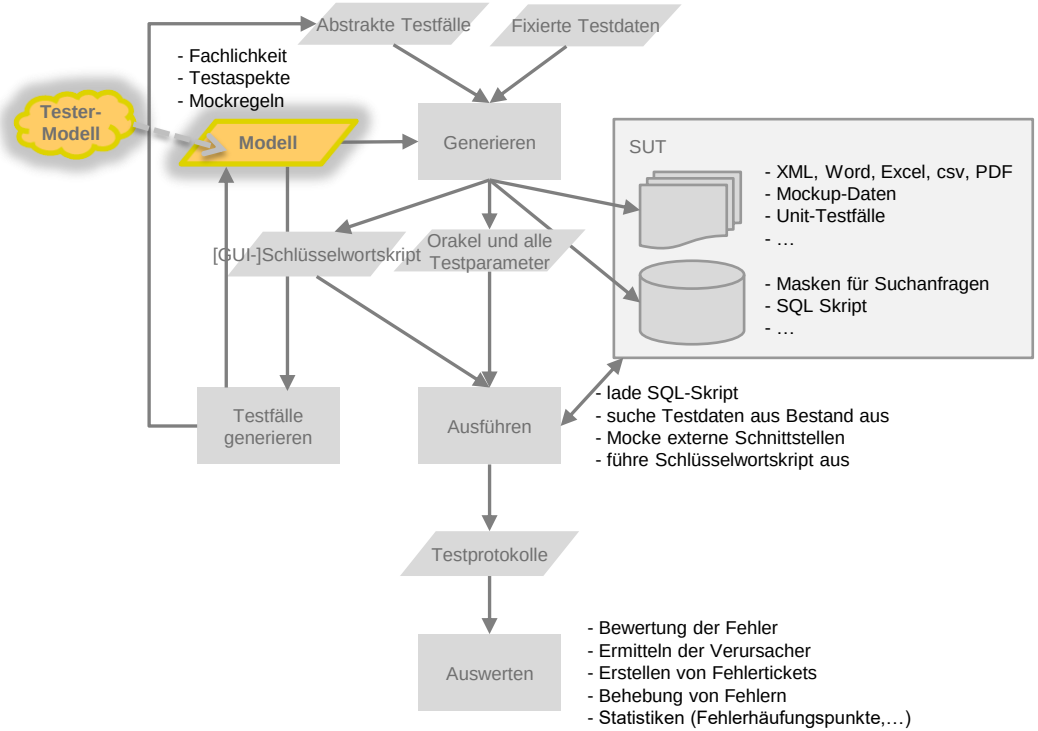
# MBT entlang Test-Reifegraden

initial

Kontrolliert

Effizient

Optimierend



# Erfahrungen im agilen Umfeld

- ✓ Testdatenmanagement: 50% Aufwand für Ermittlung der fachlichen Zusammenhänge
  - ✓ Modell: kompakt, genau und wiederverwendbar
  - ✓ Aufwand für Modellerstellung ist angemessen
- ✓ Probleme bei der Generierungen zeigen Fehler im Modell auf ⟶ schnelles Feedback ob Fachlichkeit richtig verstanden wurde
- ✓ viele Jira Tickets, welche sind relevant ⟶ Modell wird Referenz für aktuellen Stand der Anforderungen
- ✓ häufige Änderung der Fachlichkeit/Fixtures ⟶ Modell gut wartbar durch saubere Trennung: Modell, Testfälle und Ausgabeformate
- ✓ Mockregeln, fixierte Testdaten, Testaspekte, Generierung von Suchmasken: erfolgreich berücksichtigt
- ✓ Mächtigkeit von SQL ermöglicht durchgängige Testautomatisierung
- ✓ macht Spaß

http://www.agile-software-quality-assurance.com/traditional-vs-agile-qa/

# DANKE