31.8.–3.9.2015
in Nürnberg

Herbstcampus

Wissenstransfer
par excellence

Portierung von COBOL nach Java

Wie geht's? Was geht (nicht)?
Ist es sinnvoll?

Carsten Siedentop

C/S Consulting

Wer bin ich?

- Carsten Siedentop
- Consultant und Trainer für Java, COBOL, XML, ...
- > 10 Jahre Projekterfahrung in Java
- 10 Jahre Projekterfahrung in COBOL
- 17 Jahre Schulungserfahrung
- Softwareprodukte
  - PDFUnit
  - PDFMonitor
- COBOL-Buch (in Arbeit)

COBOL/Java Portierung - Agenda

- Warum ist das Thema für viele Unternehmen relevant

- Kurzer Blick auf COBOL
- Gegenüberstellung COBOL/Java
- Gegenüberstellung COBOL-Entwickler/Java-Entwickler
- Automatische Portierung
- Manuelle Portierung
- Schafft Java die Performance von COBOL
- Vorgehensweise
- Wirtschaftliche Betrachtung

- Fazit
- Fragen mit Diskussion

Wie relevant ist das Thema für Unternehmen?

Warum weg von COBOL?

- Es gehen demnächst viele Entwickler in Rente
- Es gibt zu wenig Nachwuchs für COBOL
- Viele COBOL-Systeme sind unternehmenskritisch.
- An Universitäten, Fachhochschulen wird COBOL kaum unterrichtet
- Zu wenig freie Trainer für eine zeitnahe Ausbildung
- Neulinge müssen auch CICS, ISPF, TSO etc. lernen
- Junge Java-Entwickler zieht es eher zu Java, JavaScript, Mobil-App's und Embedded Systems

Alles Gespenster?

Kurzer Blick auf COBOL

COBOL-Steckbrief (1/2)

- Geboren 1959

- Ausgangssituation (<1959):
  - Herstellerspezifische Programmiersprachen
  - FORTRAN (1958), hardware-unabhängig, aber für kaufmännische Zwecke nicht geeignet

- Ziel:
  - Sprache für kaufmännische Zwecke
  - maschinenunabhängig
  - eng an die englische Ungangssprache angelehnt

COBOL-Steckbrief (2/2)

- Hardware damals (1959):
  - kein Bildschirm
  - propietäre Tastaturen
  - keine Festplatten
  - Lochkarten, Lochstreifen

- Hauptspeicher, Prozessor (1959):
  - Single-Task Rechner
  - Memory im Kilo-Byte-Bereich
  - Taktung im Kilo-Hertz-Bereich

- COBOL-Standards: 1960, 1961, 1968, 1974, **1985**, 2002, 2014

COBOL heute

- Wenige Compiler-Hersteller:
  - IBM
  - Fujitsu-Siemens
  - Micro Focus Visual Cobol
  - ACUCOBOL von AcuCorp
  - ...
  - GnuCOBOL

- Begrenzter Kundenkreis:
  - Banken, Versicherungen
  - Großindustrie, Großhandel
  - Öffentlicher Dienst

Warum eigentlich weg von COBOL?

Gegenüberstellung COBOL - Java

Gegenüberstellung COBOL - Java (1)

	COBOL	Java
Synaxkonzept	englische Umgangssprache	formale Sprache
Schlüsselwörter	~500	50, case sensitive
Funktionen	~30	>100000
Speicher	direkter Speicherzugriff	kein direkter Zugriff
Scope von Variablen	global	private, public, ...
Parameterübergabe	by reference by value	by reference (objects) by value (primitive types)
Typische Größe	?	100 - 1000 Zeilen/Klasse

Die Aussagen spiegeln gängige Praxis wider

Gegenüberstellung COBOL - Java (1)

	COBOL	Java
Synaxkonzept	englische Umgangssprache	formale Sprache
Schlüsselwörter	~500	50, case sensitive
Funktionen	~30	>100000
Speicher	direkter Speicherzugriff	kein direkter Zugriff
Scope von Variablen	global	private, public, ...
Parameterübergabe	by reference by value	by reference (objects) by value (primitive types)
Typische Größe	500 - 50000 Zeilen/Programm	100 - 1000 Zeilen/Klasse

Die Aussagen spiegeln gängige Praxis wider

Gegenüberstellung COBOL - Java (2)

	COBOL	Java
Speicherverwaltung	kein dynamischer Speicher (seit 2002 auch dynamisch)	dynamischer Speicher => GC
Codepage (Runtime)	8-Bit ASCII, EBCDIC (seit 2002 UTF-8 + UTF-16)	UTF-16
Hardware	abhängig	unabhängig
Tools & Prozesse	Manuell dominiert	Unittests Continuous Integration

Seltene Ausnahmen bleiben unberücksichtigt

COBOL - anders, als Java-Entwickler es erwarten

	COBOL	Java
Zuweisung	move 'abc' to varX-10 move 'abc' to varX-2 move '123' to varNum	
Bedingungen	If 'abc' = 'abc '	
Schleifen	Abbruchbedingung	Laufbedingung
Arrays	Index von 1 ... n	Index von 0 ... n-1
Mathematik	Rechenergebnisse sind von Runtime-Optionen abhängig	

COBOL - anders, als Java-Entwickler es erwarten

	COBOL	Java
Zuweisung	move 'abc' to varX-10 Längenergänzung	keine Längenergänzung
	move 'abc' to varX-2 Truncation	keine Truncation
	move '123' to varNum Typkonvertierung	andere Typkonvertierungen
Bedingungen	If 'abc' = 'abc ' Längenergänzung	
Schleifen	Abbruchbedingung	Laufbedingung
Arrays	Index von 1 ... n	Index von 0 ... n-1
Mathematik	Rechenergebnisse sind von Runtime-Optionen abhängig	

Tests unabdingbar!

COBOL-Features ohne Entsprechung in Java

- Alles, was direkt mit Speicher zu tun hat
  - Pointer
  - REDEFINES
- 88-er Bedingungen
- EVALUATE
- Eigene CLASS Definitionen
- MOVE CORRESPONDING
- Der Punkt als Abschluss eines "Hauptsatzes"
- Aufruf von Unterprogrammen anderer Programmiersprachen

Kompatibilität zwischen COBOL und Java

• Datentypen	inkompatibel
• Arrays	inkompatibel
• Zuweisungen	inkompatibel
• Bedingungen	inkompatibel
• Schleifen	weitgehend kompatibel
• Rechnen	inkompatibel
• Zeichenkettenverarbeitung	inkompatibel
• Datum- und Zeitbearbeitung	inkompatibel
• Dateiverarbeitung	inkompatibel
• Graphische Oberflächen	inkompatibel
• Datenbankzugriffe	inkompatibel

Gegenüberstellung

COBOL-Entwickler / Java-Entwickler

COBOL/Java - unterschiedliches Arbeiten

- In Java
  - Für "jedes" Problem gibt es fertige Funktionen (>1000000), Logging, Validation, XML, Reporting, PDF-Erstellung, ...
  - Viele Design-Patterns verfügbar
  - Flexible Algorithmen dank dynamischer Speicherallokierung
  - Änderungen an bestehendem Code sind gut angesehen. Clean Code, Refactoring,
  - Continuous Integration, Nightly Build
- In COBOL
  - Lösungen werden "alle" selber entwickelt
  - Änderungen nur, wenn fachlich begründet
  - CI ist nicht bekannt oder wird als unmöglich angesehen

COBOL/Java - unterschiedlicher Arbeitsrhythmus

	Java	COBOL Großrechner
Zeit zwischen Codierung und Compile-Ergebnis	~1 Sekunde	~ 30 Sekunden
Compile-Vorgänge / Stunde	~ 30	< 10
Zeit zwischen Codierung und Testergebnis	< 10 Sekunden (lokale Unittests)	1 Stunde bis 4 Wochen

COBOL/Java - unterschiedliche Tools

	Java	COBOL Großrechner
Entwicklungsumgebung	sehr mächtig (Eclipse, ...)	sehr schwach (ISPF, ...)
Memory-Management	dynamischer Speicher	Working-Storage
Versionsverwaltung	mächtig (Git, SVN)	schwach (SCLM)
Build-Prozess	Continuous Integration	-
Test-Tools	JUnit, Selenium, ...	-

Automatische Portierung

Kommerzielle Anbieter (Auswahl)

- Kommerzielle Anbieter
  - EasiRun Europa GmbH, Deutschland, http://www.easirun.de/
  - pro et con, Deutschland, http://www.proetcon.de/
  - eranea, Schweiz, http://www.eranea.com/
  - MSS International, Schweiz, http://www.mssint.com/
  - CodeRenovator, USA, http://coderenovator.com/
  - Modern Systems, USA http://modernsystems.com/
  - Semantic Designs, USA, http://semanticdesigns.com/

Open Source Tools (Auswahl)

- Open Source Projekte

  - Naca
  - RES
  - CB2xml

Diese werden nachfolgend dargestellt

  - CB2java
  - JRecord
  - RecordEditor
  - ProtoBufEditor
  - FFRecord
  - OpenFileAid
  - Copybook to XML
  - JRecordBind
  - reCsvEditor

Beispielprogramm 'Zinseszins'

Datentypen

Formatierte Zahl

Zuweisung

Rechnen und Runden

Rechengenauigkeit?

Formatierung durch Zuweisung

```
data division.
working-storage section.
*-----------------------
01 zinsSatz           pic 9(02)v9(03)  value zero.
01 basisBetrag        pic 9(05)v9(02)  value zero.
01 anzahlJahre        pic 99           value zero.
01 endBetrag          pic 9(06)v9(02)  value zero.
01 endBetragFormatted pic Z(05)9.9(02) value space.

*****************************************************************
procedure division.
main section.
*-------------
    display 'Verzinsung eines Anfangsbetrags mit Zinseszins.'
    move 12 to anzahlJahre
    move 1.75 to zinsSatz
    move 50000 to basisBetrag
    compute endBetrag rounded =
                basisBetrag * ((1 + zinsSatz / 100) ** anzahlJahre)
    move endBetrag to endBetragFormatted

    display 'Basisbetrag_: ' basisBetrag
    display 'ZinsSatz____: ' zinsSatz
    display 'Anzahl Jahre: ' anzahlJahre
    display 'Endbetrag___: ' endBetragFormatted

    goback
    .
```

Example01.cob

Naca (New architecture for core applications)

Naca - Steckbrief (August 2015)

- Name: Naca (New Architecture for Core Applications)
- Website: http://code.google.com/p/naca/
- Ursprung: Internes Projekt bei "Publicitas"
- Projekt: 2003 - 2007
- Letzte Version: 1.2.0.2, Januar 2010
- Weiterentwicklung: nein
- Treffer bei Google: 4280

"Naca implements a Cobol to Java transcoder engine and provides a runtime for the generated Code."

Unternehmensgründung aus dem Projekt:
eranea, Schweiz, http://www.eranea.com

Naca - Features

- Portiert COBOL '85 nach Java:
- Setzt CICS-Programme um, einschließlich Masken
- Unterstützt Dateizugriffe zur Laufzeit (ASCII + EBCDIC)
- Führt Datenbankzugriffe (DB2) zur Laufzeit durch
- Das Laufzeitsystem kümmert sich auch um Caching

Naca - Datendeklaration

COBOL

```
01 zinsSatz            pic 9(02)v9(03)  value zero.
01 basisBetrag         pic 9(05)v9(02)  value zero.
01 anzahlJahre         pic 99           value zero.
01 endBetrag           pic 9(06)v9(02)  value zero.
01 endBetragFormatiert pic Z(05)9.9(02) value space.
```

Naca

```
Var zinssatz = declare.level(1).pic9(2, 3).valueZero().var();
Var basisbetrag = declare.level(1).pic9(5, 2).valueZero().var();
Var anzahljahre = declare.level(1).pic9(2).valueZero().var();
Var endbetrag = declare.level(1).pic9(6, 2).valueZero().var();
Var endbetragformatiert =
    declare.level(1).pic("ZZZZZ9.99").valueSpaces().var();
```

..naca..EXAMPLE01.java

Naca - prozeduraler Code

COBOL

```
move 12 to anzahlJahre
move 1.75 to zinsSatz
move 50000 to basisBetrag
compute endBetrag rounded = basisBetrag * ((1 + zinsSatz/100) ** anzahlJahre)
move endBetrag to endBetragFormatiert
display "Endbetrag___: " endBetragFormatiert
```

Naca

```
public class Example01 extends OnlineProgram {

  public void main() {
    move(12, anzahljahre);
    move("1.75", zinssatz);
    move(50000, basisbetrag);
    computeRounded(multiply(basisbetrag, power(add(1, divide(zinssatz, 100)),
                   anzahljahre)), endbetrag);
    move(endbetrag, endbetragformatiert);
    display("Endbetrag___: " + val(endbetragformatiert));
    exitProgram();
  }
}
```

Dateiname: EXAMPLE01.java

Falscher Datentyp für power(..). Compile error.

Naca - prozeduraler Code, nicht immer verständlich

```
public void counttrailingspaces() {
  compute(lengthOf(inputdata),lengthtrailingspaces);
  for (move(lengthtrailingspaces,lengthtrailingspaces);
    isSpace(inputdata.subString(lengthtrailingspaces, 1));
    dec(lengthtrailingspaces)) {
  }
}
```

COBOL

nicht immer leicht verständlich

gemeint ist:

```
public void counttrailingspaces() {
  compute(lengthOf(inputdata),lengthtrailingspaces);
  while (isSpace(inputdata.subString(lengthtrailingspaces, 1)) {
    dec(lengthtrailingspaces));
  }
}
```

Naca

Naca - Subjektive Bewertung

- Pro
  - Generierter Code ist für COBOL-Entwickler zu verstehen
  - Relative gut lesbarer Code der Procedure Division
- Con
  - Code der Data Division ist für Java-Entwickler ungewohnt
  - Kompliziertes Klassensystem, schwer zu debuggen,
  - Komplexe Runtime-Umgebung, selbst ein HelloWorld Programm benötigt eine Datenbank zur Laufzeit.
  - Nur ein Teil der COBOL-Syntax '85 wird umgesetzt
  - Unzureichende Dokumentation

RES (OpenCobol2Java)

RES (OpenCobol2Java) - Steckbrief (August 2015)

- Name: RES, OpenCobol2Java
- Website: http://sourceforge.net/projects/opencobol2java/
- Ursprung, Autor: Venkataramani Krishnamurthy
- Beginn: 2009
- Letzte Version: 2.0 Alpha, August 2010
- Weiterentwicklung: nein
- Treffer bei Google: 571

"RES is a pure Java based Cobol to Java translator. It creates maintainable Java code for a subset of VS Cobol syntax."

RES (OpenCobol2Java) - Features

- Portiert wesentliche Syntaxelemente von Cobol-VS
- Dateizugriffe
- Datenbankzugriffe

RES - Datendeklaration

COBOL

```
01 zinsSatz             pic 9(02)v9(03)  value zero.
01 basisBetrag          pic 9(05)v9(02)  value zero.
01 anzahlJahre          pic 99           value zero.
01 endBetrag            pic 9(06)v9(02)  value zero.
01 endBetragFormatiert  pic Z(05)9.9(02) value space.
```

RES

```
public class Example01 extends Program {

  public BigDecimal getZinssatz() {
    return super.getDisplayBigDecimal(0, 5, 3, false, false, false);
  }
  public String getZinssatzAsString() {
    return super.toString(0, 5);
  }
  public void setZinssatz(BigDecimal val) {
    super.setDisplayBigDecimal(0, 5, val, 3, false, false, false);
  }
  public void setZinssatz(String val) {
    super.valueOf(0, 5, val, true);
  }  ...
```

..res..Example01.java

RES - Prozeduraler Code

COBOL

```
move 12 to anzahlJahre
move 1.75 to zinsSatz
move 50000 to basisBetrag
compute endBetrag rounded = basisBetrag * ((1 + zinsSatz/100) ** anzahlJahre)
move endBetrag to endBetragFormatiert
display "Endbetrag___: " endBetragFormatiert
```

RES

```
public CobolMethod run() {
  setAnzahljahre(12);
  setZinssatz(new BigDecimal("1.75"));
  setBasisbetrag(new BigDecimal(50000));
  setEndbetrag(_Math.multiply(getBasisbetrag(),
     (_Math.pow((_Math.add(1, _Math.divide(getZinssatz(), 100))),
     getAnzahljahre()))));
  setEndbetragformatiert(__justifyRight(
     endbetragformatiertFmt_.format(getEndbetrag()), 9));
  Console.println("Endbetrag___: " + getString(getEndbetragformatiert(), 9));
  return super.run();
}
```

..res..Example01.java

Das Ergebnis ist nicht gerundet!

RES - Beispiel 'Normalize', komplizierter Code

```
Section compressmultiplespaces = new Section(this) {
  public CobolMethod run() {
      setCurrentposition(1);
      while (getCurrentposition() < getLengthtrailingspaces()) {
        if ((Compare.spaces(new CobolString(getInputdata()).substring(
            getCurrentposition() - 1, getCurrentposition() + 0)) == 0
            && Compare.spaces(new CobolString(getInputdata())
               .substring(getCurrentposition() - 1 - 1,
               getCurrentposition() - 1 + 0)) == 0)) {
          setInputdata(new CobolString(getInputdata()).refMod(
              new CobolString(getInputdata())
                  .substring( getCurrentposition() - 1),
                  getCurrentposition() - 1 - 1));
          setLengthtrailingspaces(getLengthtrailingspaces() - 1);
        } else {
          setCurrentposition(1 + getCurrentposition());
        }
      }
      return super.run();
    }
  };
```

RES

generierter Syntaxfehler

18 von 129 Zeilen

.res...Normalize.java

Zum Vergleich ...

... die handgeschriebene Lösung:

```
private static String collapseMultipleWhitespaces(String string) {
  return string.replaceAll("\\p{IsSpace}+", " ");
}
```

NormalizerOO.java

RES - Subjektive Bewertung

- Pro
  - Einfach zu bedienen
  - Knappe, aber ausreichende Dokumentation
- Con
  - Der aus der Data Division erzeugte Code ist nicht clean
  - Der aus der Procedure Division erzeugter Code ist schwierig
  - Generierter Code erzeugt Compile-Fehler und NPE
  - Nur ein Teil der COBOL-Syntax '85 wird umgesetzt

Cb2xml

Cb2xml - Steckbrief (August 2015)

- Name: Cb2xml
- Website: http://sourceforge.net/projects/cb2xml/
- Ursprung, Autor: Bruce Martin, Peter Thomas, J.F. Gagnon
- Beginn: 2004
- Letzte Version: 0.95.2, July 2015
- Weiterentwicklung: ja
- Treffer bei Google: 2280

"CB2XML (CopyBook to XML) is a COBOL CopyBook to XML converter written in Java."

Cb2xml - Beispiel 88-Bedingungsnamen

COBOL

```
01 age pic 9(03).
  88 isChild value  0 thru 13.
  88 isTeen  value 14, 15, 16, 17.
  88 isAdult value  0 thru 999.
```

Example88.cpy

Cb2xml - Beispiel 88-Bedingungsnamen

RES

```
<copybook filename="Example88.cpy">
  <item level="01" name="age" numeric="true" picture="9(03)"
        position="1" display-length="3" storage-length="3"
  >
    <condition name="isChild">
      <condition through="13" value="0" />
    </condition>
    <condition name="isTeen">
      <condition value="14" />
      <condition value="15" />
      <condition value="16" />
      <condition value="17" />
    </condition>
    <condition name="isAdult">
      <condition through="999" value="0" />
    </condition>
  </item>
</copybook>
```

Example88.cpy.xml

Cb2xml - Beispiel Arrays (Tabellen)

COBOL

```
01 week.
  05 dayField pic X(10) occurs 7.
01 dayFieldIndex pic 9(01).

01 twoDimensionalTable.
  10 dimension-1 occurs 3.
    15 dimension-2 occurs 3.
      20 varAlphanumeric pic X(02).
      20 varNumeric      pic 9(02).

01 indexDimension1 pic 9(01).
01 indexDimension2 pic 9(01).
```

Example07.cpy

Cb2xml - Beispiel Arrays (Tabellen)

RES

```
<copybook filename="Example07.cpy">
  <item level="01" name="week"
        position="1" display-length="70" storage-length="70" >
    <item level="05" name="dayField" occurs="7" picture="X(10)"
          position="1" display-length="10" storage-length="10" />
  </item>
  <item level="01" name="dayFieldIndex" numeric="true" picture="9(01)"
        position="1" display-length="1" storage-length="1" />
  <item level="01" name="twoDimensionalTable"
        position="1" display-length="36" storage-length="36" >
    <item level="10" name="dimension-1" occurs="3"
          position="1" display-length="12" storage-length="12" >
      <item level="15" name="dimension-2" occurs="3"
            position="1" display-length="4" storage-length="4" >
        <item level="20" name="varAlphanumeric" picture="X(02)"
              position="1" display-length="2" storage-length="2" />
        <item level="20" name="varNumeric" numeric="true" picture="9(02)"
              position="3" display-length="2" storage-length="2" />
      </item>
    </item>
  </item>
  <item level="01" name="indexDimension1" numeric="true" picture="9(01)"
        position="1" display-length="1" storage-length="1" />
  <item level="01" name="indexDimension2" numeric="true" picture="9(01)"
        position="1" display-length="1" storage-length="1" />
</copybook>
```

Example07.cpy.xml

Cb2xml - Verschiedene Datentypen

COBOL

```
01 num01   pic 9(04).
01 num02   pic 9(12)v9(06).
01 num03   pic s9(09)v99.

01 num04   pic 9(04) binary.
01 num05   pic 9(07) binary.

01 num07   pic s9(09)v99 binary.
01 num08   pic s9(09)v99 packed-decimal.

01 num10   comp-2.
```

CobolDataTypes.cpy

Cb2xml - Verschiedene Datentypen

RES

```
<copybook filename="CobolDataTypes.cpy">
  <item level="01" name="num01" numeric="true" picture="9(04)"
        position="1" display-length="4" storage-length="4" />
  <item level="01" name="num02" numeric="true" picture="9(12)v9(06)"
        position="1" display-length="18" storage-length="18"
        scale="6" />
  <item level="01" name="num03" numeric="true" picture="s9(09)v99"
        position="1" display-length="11" storage-length="11"
        scale="2" signed="true" />
  <item level="01" name="num04" numeric="true" picture="9(04)"
        position="1" display-length="4" storage-length="2"
        usage="binary" />
  <item level="01" name="num05" numeric="true" picture="9(07)"
        position="1" display-length="7" storage-length="4"
        usage="binary" />
  <item level="01" name="num07" numeric="true" picture="s9(09)v99"
        position="1" display-length="11" storage-length="8"
        scale="2" signed="true" usage="binary" />
  <item level="01" name="num08" numeric="true" picture="s9(09)v99"
        position="1" display-length="11" storage-length="6"
        scale="2" signed="true" usage="packed-decimal" />
  <item level="01" name="num10" numeric="true"
        position="1" display-length="18" storage-length="8"
        usage="computational-2" />
</copybook>
```

CobolDataTypes.cpy.xml

Cb2xml - Subjektive Bewertung

- Pro
  - Einfach auszuführen
  - Gutes Ergebnis
- Con
  - Es werden nur Copy-Member bearbeitet, keine Datendeklarationen von Programmen

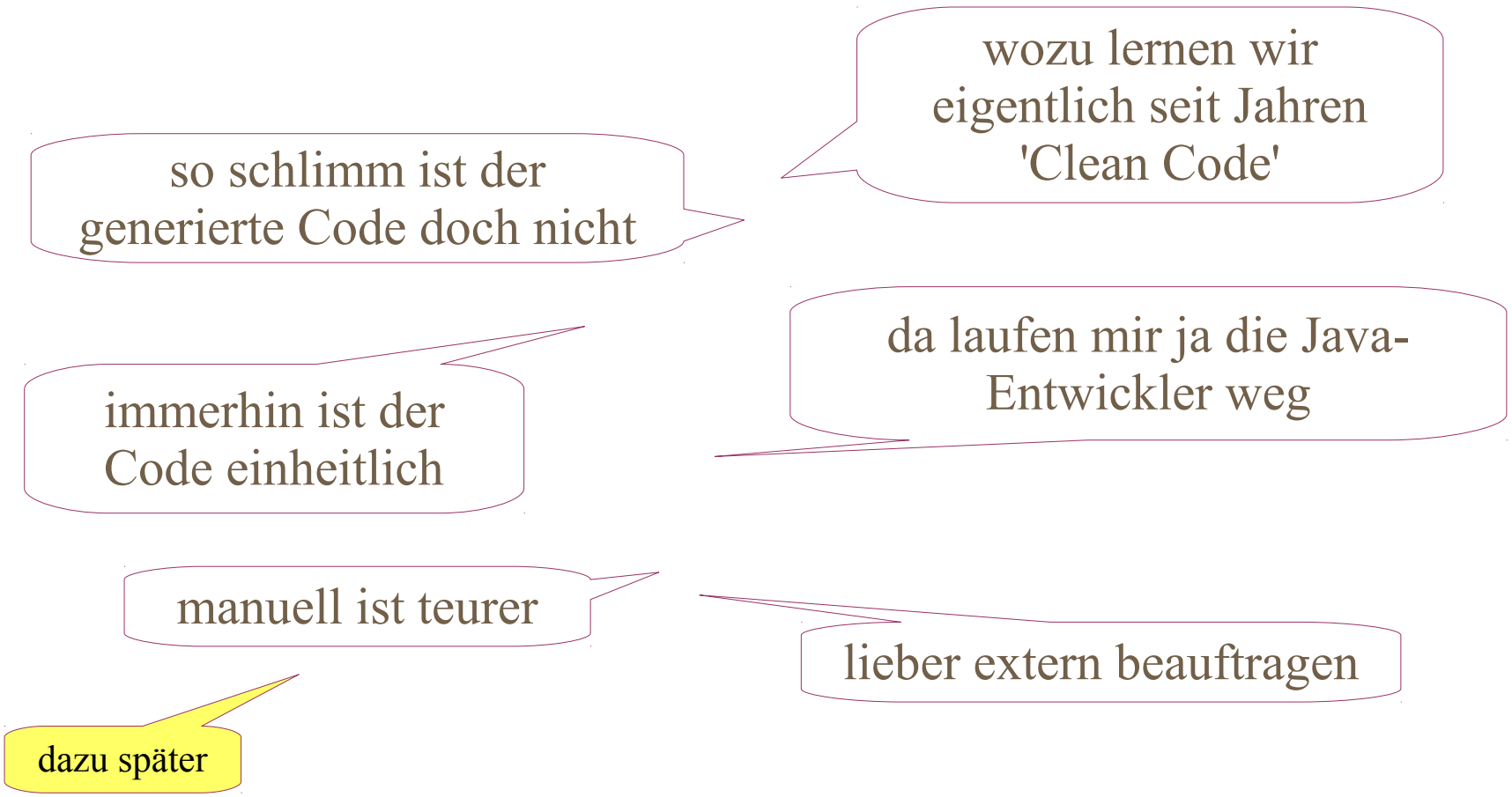
Vorteile, Nachteile automatischer Portierung

- Pro
  - Die Umstellung geht schneller, als 'manuell'
- Con
  - Fehlersuche im generierten Code ist eine Strafe
  - Prozedurale Algorithmen werden in Java abgebildet
  - Totes COBOL-Coding wird auch migriert
  - Die Ergebnisse erzeugen Compiler-Fehler, Runtime-Fehler und falsche Ergebnisse

Fazit, automatische Portierung

Wie sieht Ihr Fazit aus?

Fazit, automatische Portierung



Steckbriefe zu weiteren Tools auf separaten Folien

Manuelle Portierung

Beispielprogramm 'Zinseszins'

Datentypen

Formatierte Zahl

Zuweisung

Rechnen und Runden

Rechengenauigkeit?

Formatierung durch Zuweisung

```
data division.
working-storage section.
*-----------------------
01 zinsSatz           pic 9(02)v9(03)  value zero.
01 basisBetrag        pic 9(05)v9(02)  value zero.
01 anzahlJahre        pic 99           value zero.
01 endBetrag          pic 9(06)v9(02)  value zero.
01 endBetragFormatted pic Z(05)9.9(02) value space.

*****************************************************************
procedure division.
main section.
*-------------
    display 'Verzinsung eines Anfangsbetrags mit Zinseszins.'
    move 12 to anzahlJahre
    move 1.75 to zinsSatz
    move 50000 to basisBetrag
    compute endBetrag rounded =
                basisBetrag * ((1 + zinsSatz / 100) ** anzahlJahre)
    move endBetrag to endBetragFormatted

    display 'Basisbetrag_: ' basisBetrag
    display 'ZinsSatz____: ' zinsSatz
    display 'Anzahl Jahre: ' anzahlJahre
    display 'Endbetrag___: ' endBetragFormatted

    goback
    .
```

Example01.cob

Typisches Beispiel, Ergebnis

- Taschenrechner 61.571,965747
- COBOL, nicht gerundet 61.571,96
- COBOL, gerundet 61.571,97
- Java, gerundet (spätere Folien) 61.571,97

Picture-Klausel numerischer Datentypen

```
01 zinsSatz             pic 9(02)v9(03)  value zero.
01 basisBetrag          pic 9(05)v9(02)  value zero.
01 anzahlJahre          pic 99           value zero.
01 endBetrag            pic 9(06)v9(02)  value zero.
01 endBetragFormatiert  pic Z(05)9.9(02) value space.
```

Der Compiler berücksichtigt die Formate bei Zuweisungen, Vergleichen und Berechnungen

```
move 50000 to basisBetrag    *> Inhalt: '12345.00'
move 123456 to basisBetrag   *> Inhalt: '23456.00'

move 1.75 to zinsSatz        *> Inhalt: '1.750'
move 1.2345 to zinsSatz      *> Inhalt: '1.234'
```

Es gibt viele COBOL-Typen

Vielfalt numerischer Datentypen

```
01 num01  pic 9(04).                        *> int
01 num02  pic 9(12)v9(06).                  *> BigDecimal
01 num03  pic s9(09)v99.                    *> BigDecimal
01 num04  pic 9(04) binary.                 *> int
01 num05  pic 9(07) binary.                 *> int
01 num06  pic 9(18) binary.                 *> long
01 num07  pic s9(09)v99 binary.             *> BigDecimal
01 num08  pic s9(09)v99 packed-decimal. *> BigDecimal

*> Floating point on IBM-Host != IEEE754
01 num09  comp-1.   *> Not recommended for business programs
01 num10  comp-2.   *> Not recommended for business programs

01 num11  pic s9(35)v99 comp.               *> BigDecimal, 2002
```

Die Java-Typen decken zwar den Wertebereich ab, nicht aber das Verhalten von COBOL

Schlussfolgerung: eigene Klassen erstellen

CobolDataTypes.cob

Intelligente Java-Klasse für Zahlen

```
CobolNumber num01 = new CobolNumber("9(04)");
CobolNumber num02 = new CobolNumber("9(12)v9(06)");
CobolNumber num03 = new CobolNumber("s9(09)v99");
CobolNumber num04 = new CobolNumber("9(04) binary");

CobolNumber num07 = new CobolNumber("s9(09)v99 binary");
CobolNumber num08 = new CobolNumber("s9(9) packed-decimal");

CobolNumber num11 = new CobolNumber("s9(35)v99 comp");
```

```
CobolNumber.setValue(CobolNumber) : CobolNumber
CobolNumber.add(CobolNumber)      : CobolNumber
CobolNumber.subtract(CobolNumber) : CobolNumber

CobolNumber.format(String formatString) : String

Condition.equal(CobolNumber, CobolNumber) : boolean
Condition.lessOrEqual(CobolNumber, CobolNumber) : boolean
...
```

CobolNumber.java

Beispielprogramm 'Zinseszins' - Lösung in Java

```
CobolNumber zinsSatz          = new CobolNumber("9(02)v9(04)");
CobolNumber basisBetrag       = new CobolNumber("9(05)v9(02)");
CobolNumber anzahlJahre       = new CobolNumber("9(02)");
CobolNumber endBetrag         = new CobolNumber("9(06)v9(02)");
CobolNumberFormatted endBetragFormatted =
                                new CobolNumberFormatted("Z(06)v9(02)");

anzahlJahre.setValue(new BigDecimal("12"));
zinsSatz.setValue(new BigDecimal("1.75"));
basisBetrag.setValue(new BigDecimal("50000"));

int yearsAsInt = anzahlJahre.intValue();
endBetrag = endBetrag.setValue(zinsSatz)
                     .divide(100)
                     .add(new BigDecimal("1"))
                     .pow(yearsAsInt)
                     .multiply(basisBetrag)
                     .round();
System.out.println(endBetragFormatted.getFormatted());
```

Zuweisungen

Berechnung

Formatierung

Example01CobolNumber.java

Alphanumerische Zuweisungen

```
01 var-X-05 pic X(05).
01 var-X-10 pic X(10).

move 'abc' to var-X-05
display "var-X-05: '", var-X-05, "'"

move 'abcdefgh' to var-X-05, var-X-10
display "var-X-05: '", var-X-05, "'"
display "var-X-10: '", var-X-10, "'"
```

Sendefeld kleiner als Zielfeld

Sendefeld größer als Zielfeld

Mehrere Zielfelder

Für Zuweisungen zu numerischen Feldern gelten andere Regeln

Ausgabe:

```
var-X-05: 'abc  '
var-X-05: 'abcde'
var-X-10: 'abcdefgh  '
```

Example02.cob

Alphanumerische Zuweisungen, Lösung in Java

```
CobolString varX05 = new CobolString("X(05)");
CobolString varX10 = new CobolString("X(10)");

varX05.setValue("abc");
System.out.println("var-X-05: '" + varX05 + "'");

varX05.setValue("abcdefgh");
varX10.setValue("abcdefgh");
System.out.println("var-X-05: '" + varX05 + "'");
System.out.println("var-X-10: '" + varX10 + "'");
```

Sendefeld kleiner als Zielfeld

Sendefeld größer als Zielfeld

Ausgabe:

```
var-X-05: 'abc  '
var-X-05: 'abcde'
var-X-10: 'abcdefgh  '
```

Example02.java

Zuweisung zu Gruppen

```
01 germanDate.
  05 dd     pic 9(02).
  05 filler pic X(01).
  05 mm     pic 9(02).
  05 filler pic X(01).
  05 yyyy   pic 9(04).

move '31.12.2014' to germanDate
display 'day__: ' dd
display 'month: ' mm
display 'year_: ' yyyy
```

10 Byte

Move beginnt mit dem linken Byte

Ausgabe:

```
day__: 31
month: 12
year_: 2014
```

Example03.cob

COBOL-Gruppen werden zu Java-Klassen

```
public class GermanDate {

  private CobolString dd = new CobolString("X(02)");
  private CobolString filler1 = new CobolString("X(01)");
  private CobolString mm = new CobolString("X(02)");
  private CobolString filler2 = new CobolString("X(01)");
  private CobolString yyyy = new CobolString("X(04)");

  public void setValue(String newValue) {
    dd.setValue(newValue.substring(0, 2));
    filler1.setValue(newValue.substring(2, 3));
    mm.setValue(newValue.substring(3, 5));
    filler2.setValue(newValue.substring(5, 6));
    yyyy.setValue(newValue.substring(6, 10));
  }

}
```

GermanDate.java

Zuweisung zu Gruppen - Lösung in Java

```
public class Example03 {

  private static GermanDate germanDate = new GermanDate();

  public static void main(String[] args) {
    germanDate.setValue("31.12.2014");
    System.out.println("day__: " + germanDate.getDd());
    System.out.println("month: " + germanDate.getMm());
    System.out.println("year_: " + germanDate.getYyyy());
  }

}
```

Ausgabe:

```
day__: 31
month: 12
year_: 2014
```

Example03.java

Datentypübergreifende Zuweisung

```
01 varNumeric      pic 9(03).
01 varAlphanumeric pic X(03).

move '123' to varNumeric
move '456' to varAlphanumeric
move varAlphanumeric to varNumeric

display 'varAlphanumeric: ' varAlphanumeric
display 'varNumeric_____: ' varNumeric
```

Ausgabe:

```
varAlphanumeric: 456
varNumeric_____: 456
```

Example04.cob

Datentypübergreifende Zuweisung - Lösung in Java

```
private static CobolNumber varNumeric = new CobolNumber("9(03)");
private static CobolString varAlphanumeric = new CobolString("X(03)");

public static void main(String[] args) {
  varNumeric.setValue(123);
  varAlphanumeric.setValue("456");
  varNumeric.setValue(varAlphanumeric);
  System.out.println("varAlphanumeric: " + varAlphanumeric.getValue());
  System.out.println("varNumeric_____: " + varNumeric.getValue());
}
```

Ausgabe:

```
varAlphanumeric: 456
varNumeric_____: 456
```

Example04.java

'Normale' Bedingungen

```
01 varNumeric-4 pic 9(04).
01 varNumeric-9 pic 9(09).

move 4711 to varNumeric-4, varNumeric-9
if varNumeric-4 = varNumeric-9
  display 'field 1 is equal to field 2'

if 'abc' = 'abc    '
  display 'field 1 is equal to field 2'
```

Leading zeroes!

Trailing spaces!

Die Java-Implementierung ist einfach:

IfEqualTest.java

```
CobolNumber num1 = new CobolNumber("9(04)");
CobolNumber num2 = new CobolNumber("9(09)");
num1.setValue(new BigDecimal(4711));
num2.setValue(new BigDecimal(4711));
result1 = NumericCondition.equal(num1, num2);

result2 = AlphanumericCondition.equals("abc", "abc    ");
```

Klassenbedingungen, 'class conditions'

```
move 4711 to varNumeric
if varNumeric is positive ...                *> true

move '12345' to varAlphanumeric-X5
if varAlphanumeric-X5 is numeric ...         *> true

move '12345' to varAlphanumeric-X10
if varAlphanumeric-X10 is numeric ...        *> false

move 'foo bar' to varAlphanumeric-X10
if varAlphanumeric-X10 is alphabetic ...     *> true

move 'foo 123' to varAlphanumeric-X10
if varAlphanumeric-X10 is alphabetic ...     *> false
```

Trailing spaces!

Ciphers

Die Java-Implementierung ist einfach:

```
ClassCondition.isNumeric("12345     ");
ClassCondition.isAlphabetic("foo 123");
```

ClassConditionTest.java

Codepage-abhängige Vergleiche

```
if 'ABC' less 'abc'
  display ...
else
  display ...
end-if
```

- Wie lautet das Ergebnis?
  - 'ABC' ist kleiner als 'abc'
  - 'abc' ist kleiner als 'ABC'
  - 'ABC' und 'abc' sind gleichwertig

Codepage-abhängige Vergleiche

```
if 'ABC' less 'abc'
  display "'ABC' is less 'abc' in ASCII"
else
  display "'abc' is less 'ABC' in EBCDIC"
end-if
```

Beides kann richtig sein

- Auswirkung bei
  - Bedingungen <, >, less, greater, ...
  - Schleifen perform...until
  - Sortierung sort, merge

IfLessTest.java

Und was ist hiermit:

```
if 'ABC' equal 'abc'
  display "'ABC' is equal to 'abc'"
else
  display "'ABC' is not equal to 'abc'"
end-if
```

CollatingSequenceDemo.cob

Ja, COBOL kann das!

```
if 'ABC' equal 'abc'
  display "'ABC' is equal to 'abc'"
else
  display "'ABC' is not equal to 'ABC'"
end-if
```

'ABC' und 'abc' können gleichwertig sein!

Maclean
maconda
MACRO
Macromedia
MACTac

Deutsches Telefonbuch

Ausgabe:

```
'ABC' is equal to 'abc'
```

Java:
Comparator implementieren

88-er Bedingungen

```
01 age pic 9(03).
  88 isChild value  0 thru 13.
  88 isTeen  value 14, 15, 16, 17.
  88 isAdult value  0 thru 999.

move 20 to age
if isAdult
   display 'The person is 18 or elder'
```

Example88.cob

Die Java-Implementierung ist einfach:

```
Age age = new Age("9(03)");
age.setValue(20);
boolean isAdult = age.isAdult();
if (isAdult) {
   System.out.println("The person is 18 or elder.");
}
```

Example88.java

Schleifen

```
perform varying loopCounter from 1 by 1
        until loopCounter = maxCounter
  display 'loopCounter: ' loopCounter
end-perform

move zero to loopCounter
perform until loopCounter = maxCounter
  display 'loopCounter: ' loopCounter
  add 1 to loopCounter
end-perform
```

Abbruchbedingungen

Example05.cob

```
for (int loopCounter = 1; loopCounter != maxCounter; loopCounter++) {
  System.out.println("loopCounter: " + loopCounter);
}

while (loopCounter != maxCounter) {
  System.out.println("loopCounter: " + loopCounter);
  loopCounter++;
}
```

Laufbedingungen

Example05.java

Komplizierte Schleifen

```
perform varying counter1 from 1 by 1 until counter1 > 3
        after   counter2 from 1 by 1 until counter2 > 3
        after   counter3 from 1 by 1 until counter3 > 3
  display 'counter: ' counter1 '.' counter2 '.' counter3
end-perform
```

Example05.cob

Die Java-Implementierung ist einfach:

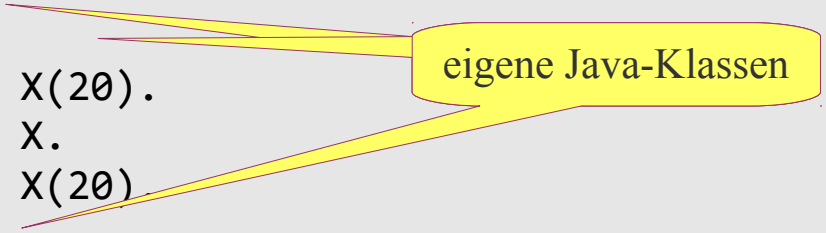
```
for (int counter1 = 1; !(counter1 > 3); counter1++) {
  for (int counter2 = 1; !(counter2 > 3); counter2++) {
    for (int counter3 = 1; !(counter3 > 3); counter3++) {
      System.out.println("counter: "
                         + counter1 + "." + counter2 + "." + counter3);
    }
  }
}
```

Example05.java

Datengruppen

```
01 customer.
  05 customerName.
    10 firstName  pic X(20).
    10 middleName pic X.
    10 lastName   pic X(20).
  05 customerAddress.
    10 street     pic X(40).
    10 zipCode    pic X(10).
    10 city       pic X(30).

move 'Carsten' to firstName
move 'Siedentop' to lastName
display customerName
```



Ausgabe:

```
Carsten      Siedentop
```

Example06.cob

Datengruppen - Lösung in Java

```
public class Example06 {

  private static Customer customer = new Customer();

  public static void main(String[] args) {
    System.out.println("Example for data groups");
    customer.setFirstName("Carsten");
    customer.setLastName("Siedentop");
    String customerName = customer.getCustomerName();
    System.out.println(customerName);
  }

}
```

Customer.java

CustomerName.java

CustomerAddress.java

Example06.java

Ausgabe:

```
Carsten      Siedentop
```

Arrays, Tabellen

```
01 week.
  05 dayField pic X(10) occurs 7.
01 dayFieldIndex pic 9(01).

move 'Monday' to dayField(1)
move 7 to dayFieldIndex
move 'Sunday' to dayField(7)
display '1. day: ' dayField(1)
display '7. day: ' dayField(7)
```

COBOL-Index von 1..n

Example07.cob

Die Java-Implementierung ist einfach:

```
Week week = new Week();
week.getDayField(1).setValue("Monday");
dayFieldIndex = 7;
week.getDayField(dayFieldIndex).setValue("Sunday");
System.out.println("1. day: " + week.getDayField(1));
System.out.println("7. day: " + week.getDayField(7));
```

könnte auch 0..(n-1) sein

Example07.java

Zeichenkettenbearbeitung in COBOL

```
UNSTRING identifier-1
    [ DELIMITED BY [ ALL ] { identifier-2 | literal-1 } [ OR [ ALL ] { identifier-3 | literal-2 } ] ... ]
    INTO { identifier-4 [ DELIMITER IN identifier-5 ] [ COUNT IN identifier-6 ] } ...
    [ WITH POINTER identifier-7 ]
    [ TALLYING IN identifier-8 ]
    [ ON OVERFLOW imperative-statement-1
      NOT ON OVERFLOW imperative-statement-2 ]
    [ END-UNSTRING ]
```

```
STRING { { identifier-1 | literal-1 } ... [ DELIMITED BY { identifier-2 | literal-2 | SIZE } ] } ...
    INTO identifier-3
    [ WITH POINTER identifier-4 ]
    [ ON OVERFLOW imperative-statement-1
      NOT ON OVERFLOW imperative-statement-2 ]
    [ END-STRING ]
```

Zeichenkettenbearbeitung in Java - alles schon fertig

StringTokenizer
DecimalFormat
String
Pattern
MessageFormat
DateFormat
StringBuffer
Regular Expression
matches()
toUpperCase()
reverse()
toLowerCase()
endsWith()
split()
append()
startsWith()
parse()
toString()
printf()
indexOf()
format()
insert()
replace()
substring()

Beispiel 'Zeichenkette normalisieren' in COBOL

```
*> countTrailingSpaces
   compute lengthTrailingSpaces = function length(inputData)
   perform varying lengthTrailingSpaces from lengthTrailingSpaces by -1
           until inputData(lengthTrailingSpaces:1) not = space
     continue
   end-perform
*> compressMultipleSpaces
   move 1 to currentPosition
   perform until currentPosition >= lengthTrailingSpaces
     if (inputData(currentPosition:1) = space and
         inputData(currentPosition - 1:1) = space)
       move inputData(currentPosition:) to inputData(currentPosition - 1:)
       subtract 1 from lengthTrailingSpaces
     else
       add 1 to currentPosition
     end-if
   end-perform
*> removeLeadingSpace
   if inputData(1:1) = space
     move inputData(2:) to inputData (1:)
   end-if
```

NormalizeDemo.cob

Beispiel 'Zeichenkette normalisieren' in Java

```java
public static String normalize(String value) {
  String trimmendValue = value.trim();
  String collapsedValue = collapseMultipleWhitespaces(trimmedValue);
  return collapsedValue;
}

private static String collapseMultipleWhitespaces(String string) {
  return string.replaceAll("\\p{IsSpace}+", " ");
}
```

NormalizerOO.java

Diese Java-Implementierung ist einfach.

NormalizerProcedural.java

- Eine manuell erstellte, prozedurale Lösung benötigt 33 Zeilen.
- Eine automatisch aus COBOL erstellte Lösung benötigt 121 Zeilen.

.res...Normalize.java

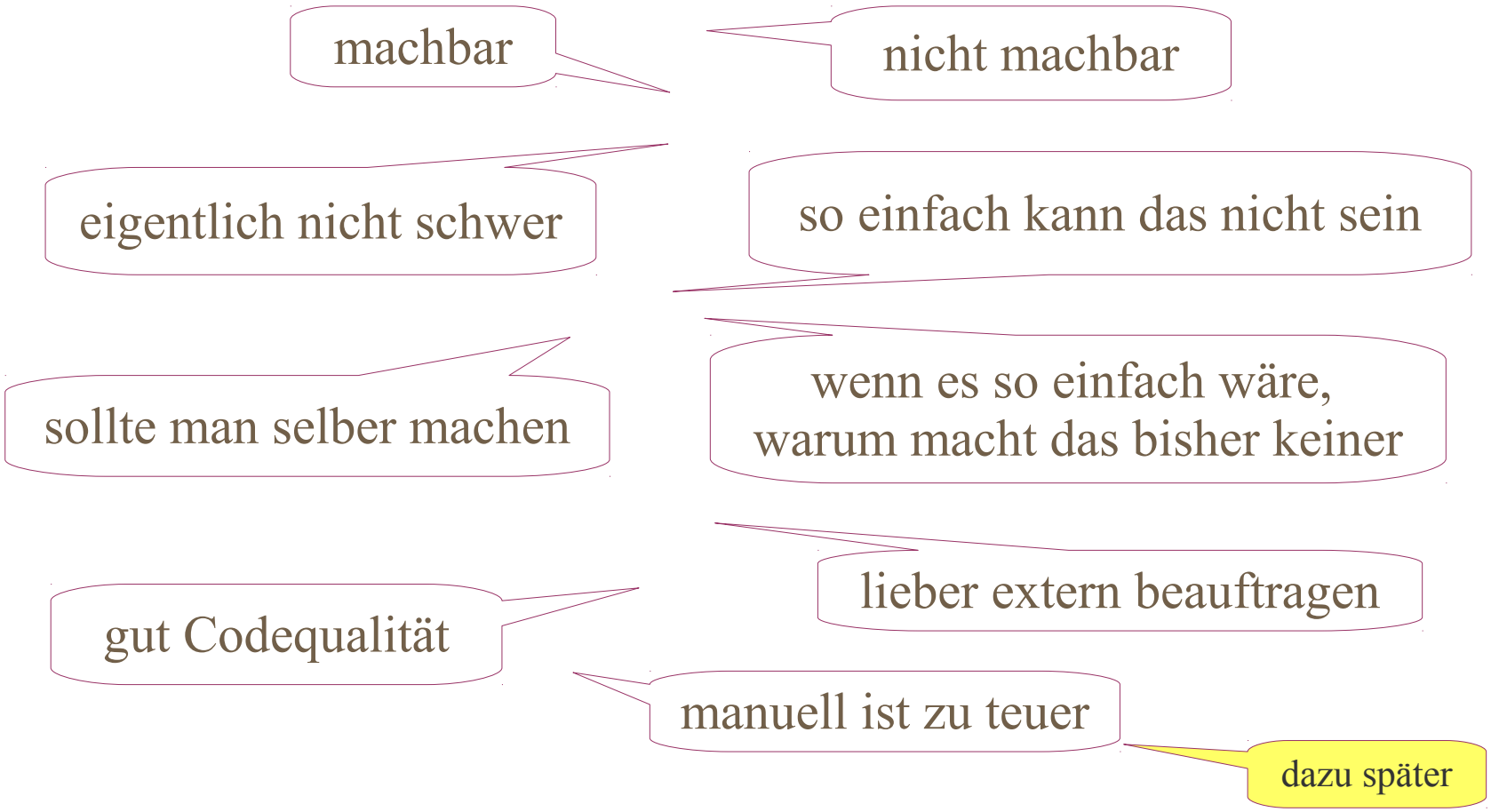
Vorteile, Nachteile manueller Portierung

- Pro
  - Gute Code-Qualität möglich
  - Code ist gut lesbar
  - Code is auch langfristig verständlich
  - Algorithmen können objektorientiert implementiert werden
  - Funktionalität etablierter Bibliotheken ist nutzbar
- Con
  - Die Umstellung dauert länger, als 'automatisch'
  - Änderungen am COBOL-Code während der Umstellung können nicht 'automatisch' in Java nachgezogen werden
  - Entwickler mit Java- und COBOL-Wissen werden benötigt

Fazit, manuelle Portierung

Wie sieht Ihr Fazit aus?

Fazit, manuelle Portierung



Schafft Java die Performance von COBOL?

Schafft Java die Performance von COBOL?

Ja.

Vorgehensweise

Migration - was zuerst

- **Batches First**
  - Datei- und DB-Zugriffe, kein Frontend
  - relativ gut testbar, da keine Benutzerinteraktion
- **Frontend First**
  - 3270 Terminals, IBM
  - Windows-GUI, Micro Focus
  - ASCII-Screen, COBOL Screen Section
  - schwierig zu testen, aber Batch-Programme bleiben stabil
- **Modul für Modul**?
  - wo sind die Grenzen eines Moduls

Oder Big-Bang?

Migration - was zuerst

- **Batches First**
  - Datei- und DB-Zugriffe, kein Frontend
  - relativ gut testbar, da keine Benutzerinteraktion
- **Frontend First**
  - 3270 Termi
  - Windows-GUI, Micro Focus
  - ASCII-Screen, COBOL Screen Section
  - schwierig zu testen, aber Batch-Programme bleiben stabil
- **Modul für Modul**?
  - wo sind die Grenzen eines Moduls

Oder besser gar nicht migrieren?

Migration - schwierige Randbedingungen

- Der COBOL-Standard ist weich.
  - Im Standard 2014: 200x 'undefined', 150x 'implementor-defined'
- COBOL-Programme enthalten sprachfremde Erweiterungen:
  - für Systemzugriffe (Memory, etc.)
  - für Zugriffe auf Datenbanken
  - für graphische Oberflächen
  - für andere nicht verfügbare Funktionen,
    z.B. Message-Queuing, XML

herstellerspezifisch

- Die Fremdsysteme bieten häufig Java-Schnittstellen:
  - MQSeries for Java, CICS Transaction Gateway, ECI
  - JPA, Hibernate, ...

Vorgehensweise - persönliche Empfehlung

- Erstellen Sie viele automatische Tests für das Altsystem
- Migrieren Sie zuerst ein Batch-Programm
- Portieren Sie die Testprogramme (automatisch) nach JUnit
- Generieren Sie die Datendeklaration automatisch
  - cb2xml nutzen, eventuell erweitern
- Schreiben Sie den prozeduralen Teil händisch
  - Keine prozeduralen Lösungen nachbauen, sondern OO
  - Patterns nutzen: Singleton, FluentBuilder, ...
  - Standardbibliotheken nutzen
  - Etablierte 3rd-Party Bibliotheken nutzen

Vorher gründlich analysieren

Wirtschaftliche Betrachtung

Bestandsaufnahme IST und SOLL

- Analysieren Sie die Personal-Situation (ehrlich)
  - Welches Know-how hat Ihr Personal?
  - Wie alt ist Ihr Personal
  - Welches Wissen geht innerhalb der kommenden 5 Jahre in Rente
  - Gibt es Kopfmonopole
- Wie lange sollen Ihre Anwendungen weiterbetrieben werden?
- Wieviele Änderungen sind in dem Zeitraum zu erwarten?
- Wollen Sie kurzfristig Geld sparen, oder langfristig?
- Welche Anwendung gefährdet Ihr Unternehmen, falls sie ausfällt?

Wirtschaftliche Betrachtung - bis zur Einführung

- Am teuersten ist das Testen (~80%)
  - Sie müssen das Altsystem (COBOL) und das neue System (Java) automatisiert testen
- Automatisiertes Testen in COBOL braucht:
  - neue Umgebungen
  - neue Tools
  - neue Prozesse
  - neue Mentalität aller Beteiligten
- Was kostet Sie der Ausfall Ihrer Anwendungen
- Welchen Nutzen bringt eine Umstellung bis zur Einführung?

Wirtschaftliche Betrachtung - nach der Einführung

- Wie gut ist der Java-Code
  - zu verstehen
  - zu debuggen
  - zu testen
  - zu erweitern
  - zu refactor'n
- Wie werden die COBOL-Entwickler langfristig eingesetzt
- Wie schnell sind neue Anforderungen umgesetzt
- Die vorhandenen Tests reduzieren die Kosten für Bug-Fixes

Kosten manueller + automatischer Portierung

	manuell	automatisiert
Erstellung Testsystem COBOL	sehr hoch	sehr hoch
Erstellung eigener Tools	gering	gering
Anpassung externer Tools	-	**mittel**
Testen des COBOL-Systems	mittel	mittel
Testen des Java-Systems	mittel	mittel
Manuelle Code-Erstellung	**hoch**	**gering**
Pflege der Tools nach der Portierung	**gering**	**mittel**

Diese Aussagen sind von vielen Faktoren abhängig.

Fazit der wirtschaftlichen Betrachtung

Wie sieht Ihr Fazit aus?

Fazit der wirtschaftlichen Betrachtung

wer soll das denn bezahlen

warum umstellen,
es läuft doch

in zwei Jahren bin ich
sowieso in Rente

das klappt nie

ich würde so ein Projekt
machen, wenn der Vorstand
grünes Licht gibt

Fazit der wirtschaftlichen Betrachtung

- Es wird in jedem Falle teuer
  - Zusätzliche Java-Entwickler beschäftigen
  - COBOL-Entwickler in Java ausbilden
  - Nach SAP auslagern
- Mit COBOL weitermachen?
  - Das verschiebt Ihr Problem lediglich um 5 - 10 Jahre
  - Frisch ausgebildete COBOL-Programmierer brauchen lange, 20 Jahre alten Code zu verstehen
  - Es gibt nicht genügend COBOL-Trainer
- Warten, bis eine Unternehmensfusion ansteht?
  - Dann können Sie die Kosten verstecken

Persönliches Fazit

Persönliche Empfehlung für eine Portierung (1/2)

- Stellen Sie ein gemischtes Team zusammen
  - COBOL-Entwickler mit Java-Kenntnissen
  - Java-Entwickler mit COBOL-Kenntnissen
  - Begeisterungsfähige Berufsanfänger und Alte Hasen
- Befreien Sie das Team vom Tagesgeschäft
- Geben Sie dem Team ein eigenes Testsystem
- Geben Sie dem Team Entscheidungsbefugnis
- Keine Zielvereinbarungen für die Teammitglieder
- Geben Sie dem Team im Testsystem vollen Zugriff auf
  - System, Berechtigungssystem, Datenbank, Tools, Internet

Persönliche Empfehlung für eine Portierung (1/2)

Fangen Sie an,

solange noch COBOL- und Fachwissen im Hause ist

Haben Sie eine Alternative?

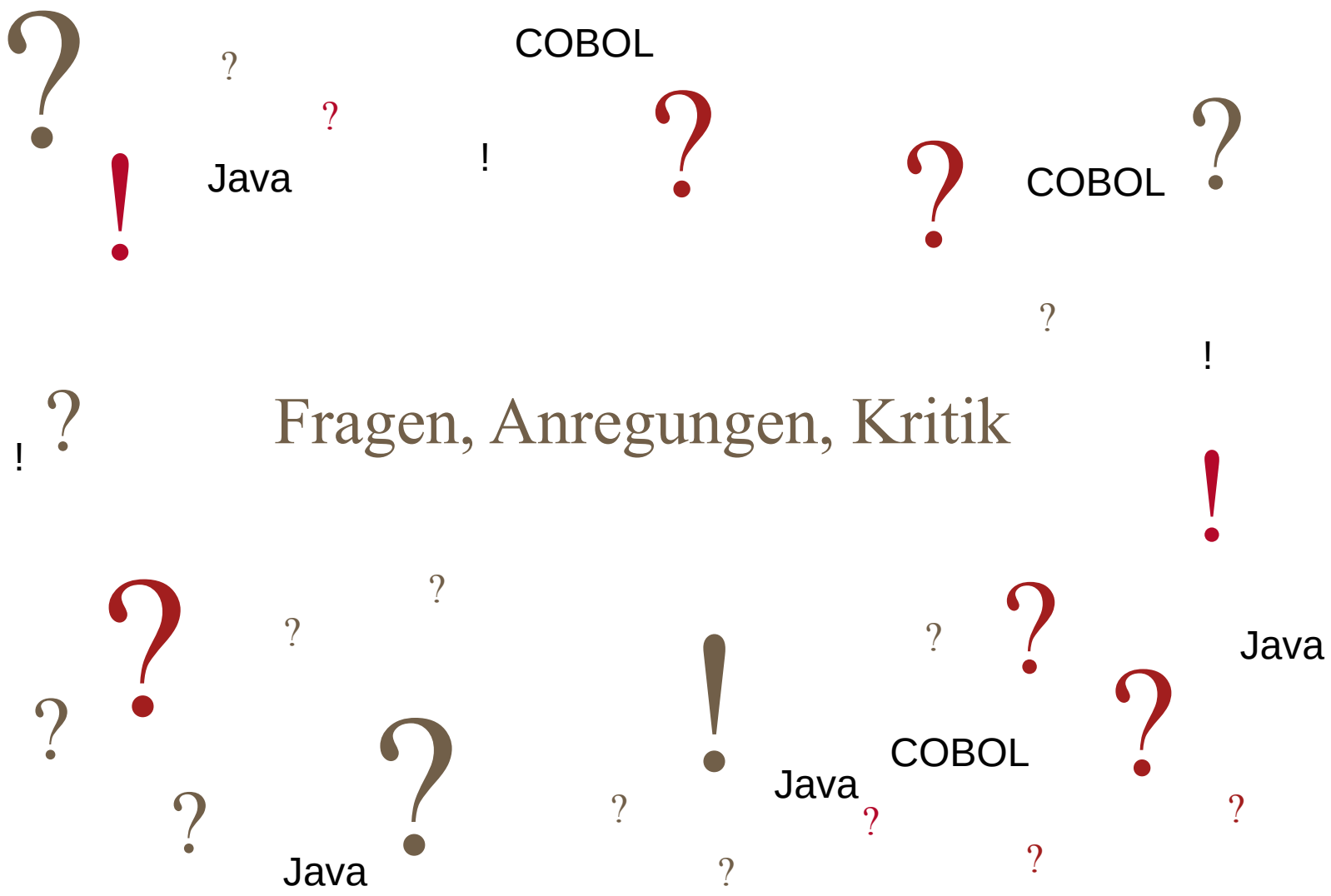
31.8.–3.9.2015
in Nürnberg

Herbstcampus

Wissenstransfer
par excellence

Viel Erfolg mit und ohne COBOL!

Carsten Siedentop


COBOL
Java
COBOL
Fragen, Anregungen, Kritik
Java
Java
COBOL
Java

31.8.–3.9.2015
in Nürnberg

Herbstcampus

Wissenstransfer
par excellence

Viel Erfolg mit und ohne COBOL!

Carsten Siedentop