Program Analysis and Transformation from a Practitioner´s POV

Roland Hildebrandt
hildebrandt@itestra.de

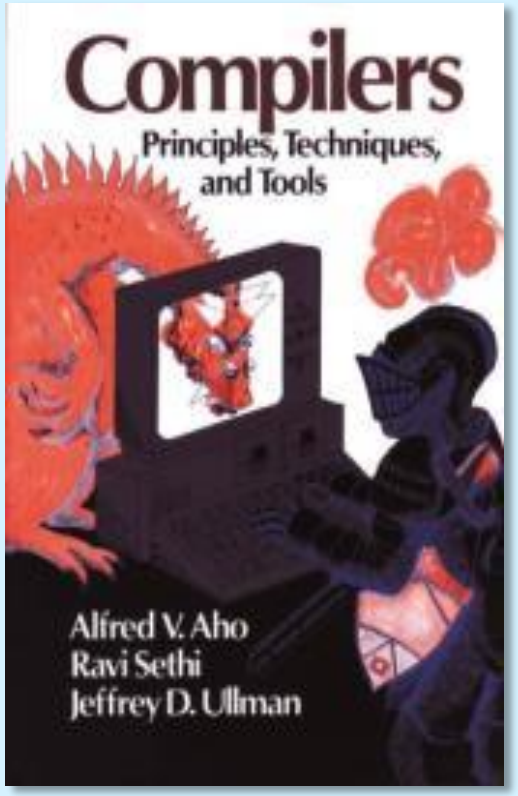
17.12.2018

itestra
be excellent

itestra
be excellent

Many CS students
attend a **compiler contruction class.**

Few of them ever **build a compiler**
in their professional life.



Agenda

itestra
be excellent

01 Background
- Business Information Systems
- itestra

02 Use Cases for Compiler Construction Methods in Practice

03 Program analysis
- Challenges, Experiences, Approaches

04 Programm transformation
- Challenges, Experiences, Approaches

Kapitel 01

Background

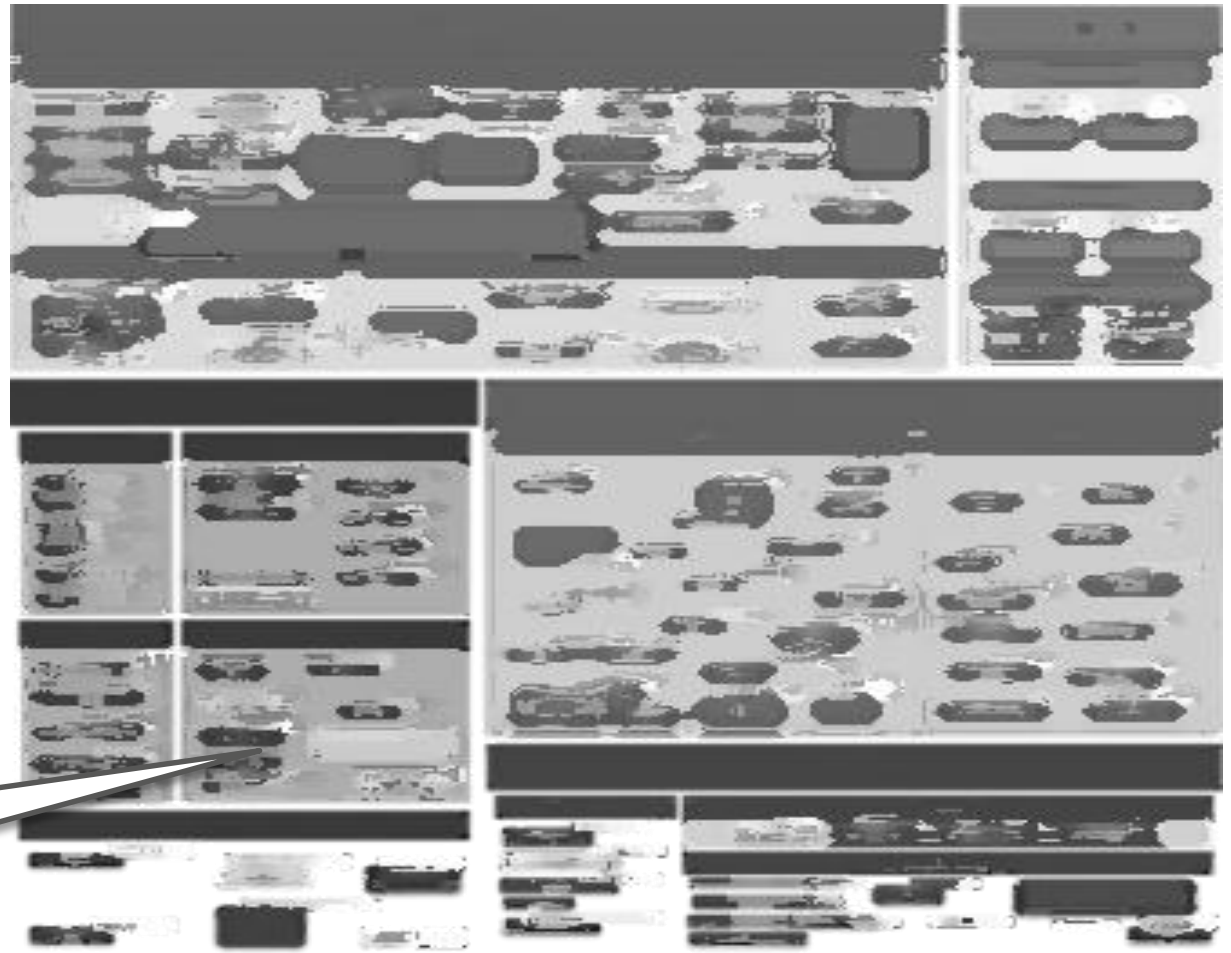
Reality in most large scale enterprises

100s or even 1.000s of “applications”

10 - 100 Mio. LoC, value of 100 – 1.000 Mio. €

often several decades old

Age & Size
mean
Success!



Your bank account is being processed here…

Typical Technologies

itestra
be excellent

Languages:

Java, JavaScript, ABAP

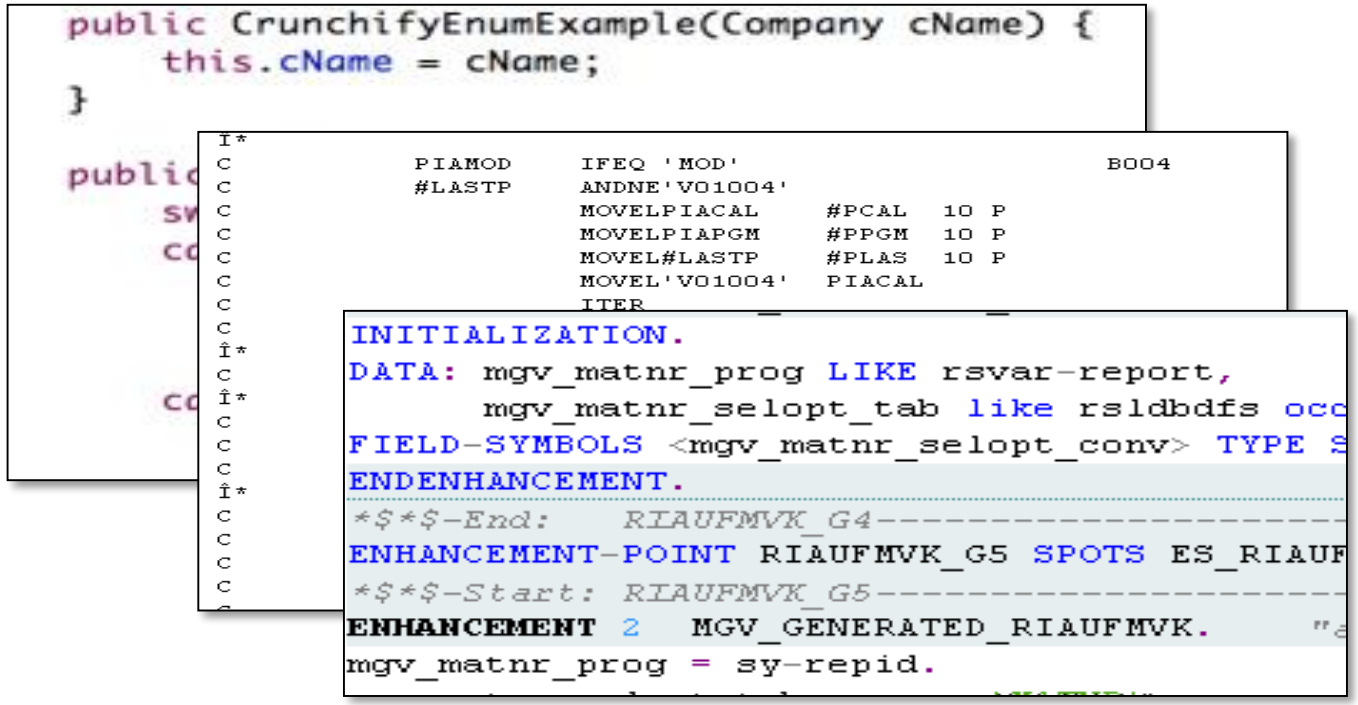
COBOL, PL/I, RPG, NATURAL, C/C++

Assembler

VaGen, DeltaGen,

SAS, Easytrieve

PowerBuilder, Gupta, Synon, …



Data: DB2, Oracle, VSAM, IDMS, ADABAS, …

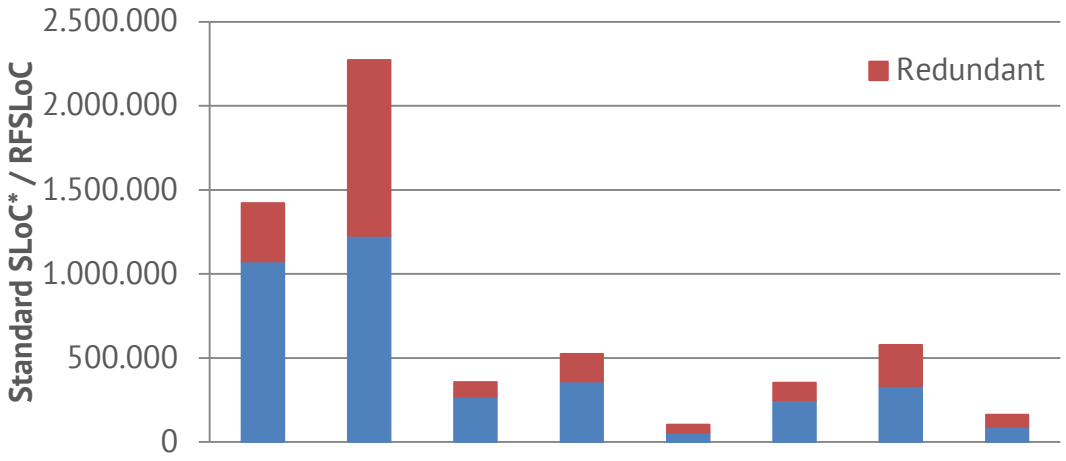
„Middleware“: CICS, IMS, MQ, …

Heterogenity is unavoidable in practice!

Long lifecycle, too big to reimplement/migrate everything every 10 years

Mergers & acquisitions (besides, heterogenity also due to hypes/personal preferences, lack of strategy, …)

Typical situation: High Redundancy



Also:

Unused, e.g. 30 % Unused DB Tables

Inappropriate implementations

No documentation, incomprehensible naming

In particular for
elder systems,
but equally existing for
newly developed Java!

Consequences e.g.: repeated / superfluous / overly expensive modifications

Business changes such as SEPA, Euro (costing Mio. of € to implement)

Technical changes such as compiler version, new framework, …

Real example: Extent attribute from 4 to 8 digits in **10.000** locations

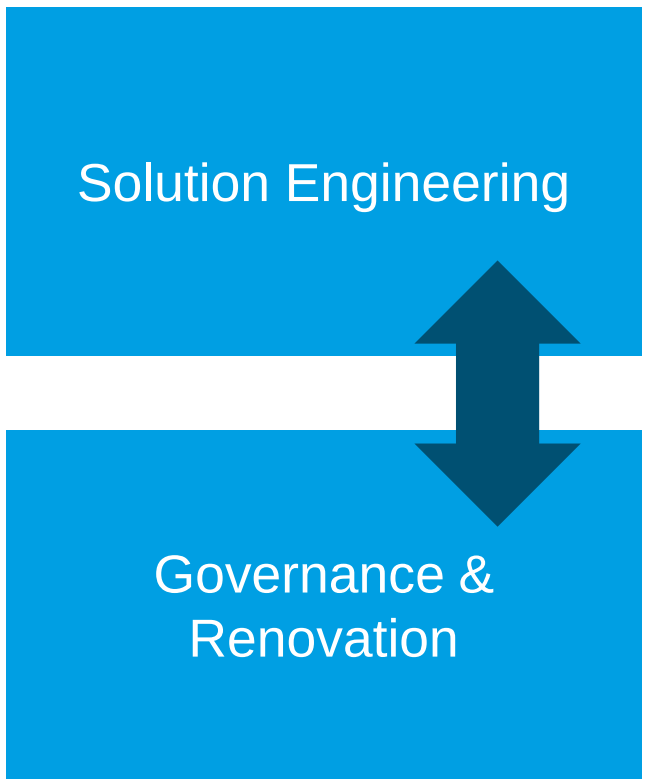
About itestra

Founded 2003,
independent financially and technologically

75 (fixed contract) employees

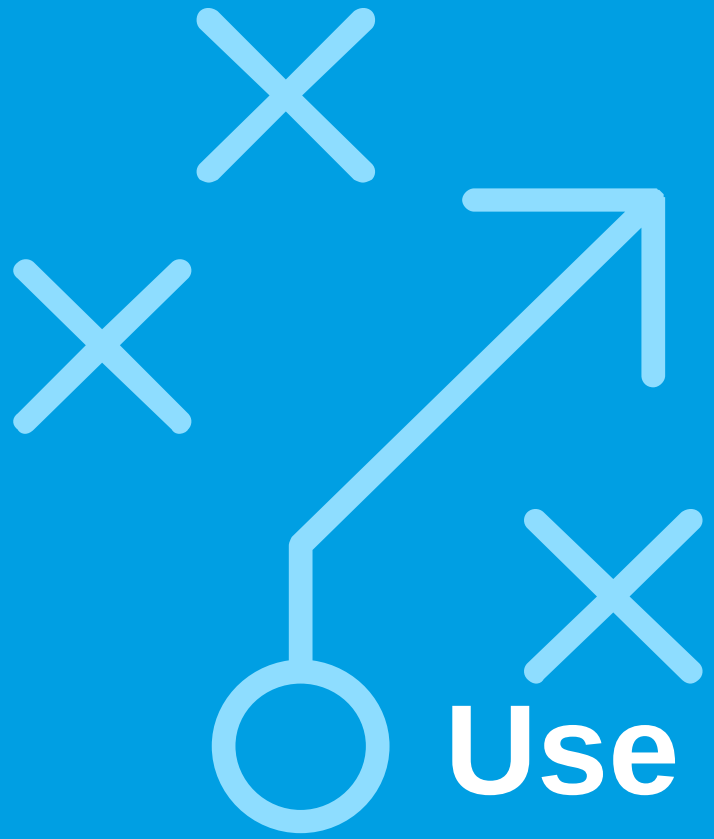
8 locations - München, **Köln**, Stuttgart, Nürnberg, Hamburg, Hannover, Madrid, Tallinn

Service portfolio



- Realisation of highly configurable IT solutions for business-critical processes

- Measurement of quality and productivity
- Software renovation and optimisation
- Reducing costs and increasing business value



itestra
be excellent

Kapitel 02

Use Cases for Compiler Construction Methods in Practice

Use Cases from a practitioner‘s POV

itestra
be excellent

Program Analysis

Identification of **redundant** or **unused** parts

Estimation of **size**, **appropriate cost** and **renovation/migration effort**

Identification of economically relevant **quality problems**

Impact analysis for changes / Support for maintenance/development staff

Program transformation

Automated **migration (?)**

Automated **adaption**, e.g.

- systematic, high-impact technical change
- compiler or framework version

Refactoring support for maintenance/development staff

Isn‘t all of this build in Eclipse/IDEA IDE?

itestra
be excellent

Often not covered (sufficiently) by existing tools:

Elder languages and technologies, Precompilers and macros

⇨ lacking modern tool support

Scripting, **modelling** and **configuration** „languages“

Batch scripts, e.g. in Shell Script, JCL

Build configuration, e.g. ANT, mainframe compile jobs

Report and document generation, e.g. Jasper, Easytrieve

Graphic and other models, e.g. UML, workflow

⇨ often overlooked, but may grow to significant size!
Similar activitites in SW lifecycle as with code!

Specific activitites not supported by common tools ⇨ **Need to create custom tools!**

A glance at the future

itestra
be excellent

Will this be important any more when all legacy systems have been shut off?

Everything from scratch every 10 years is impossible (cost & manpower!)
⇨ Need to conserve a system‘s value through renovation

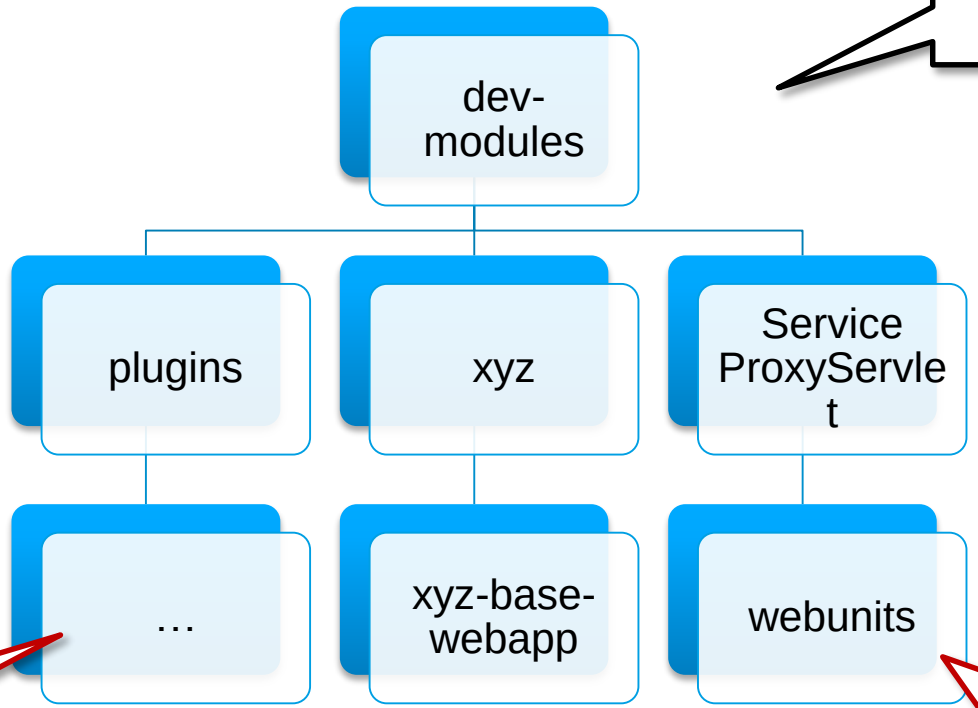
Today‘s software is tomorrow‘s legacy
(EJB 1.0, Java w/o Generics, classic ABAP, PHP3, …)

Software Engineering is Dead, Buy before Make
i.e. Customizing & Configuration of COTS is increasing

Product- und domain specific languages
⇨ large amount of „source“ in proprietary format

Example: Maven POM analysis

Maven build configuration
of a larger insurance system
(several dozen build modules)



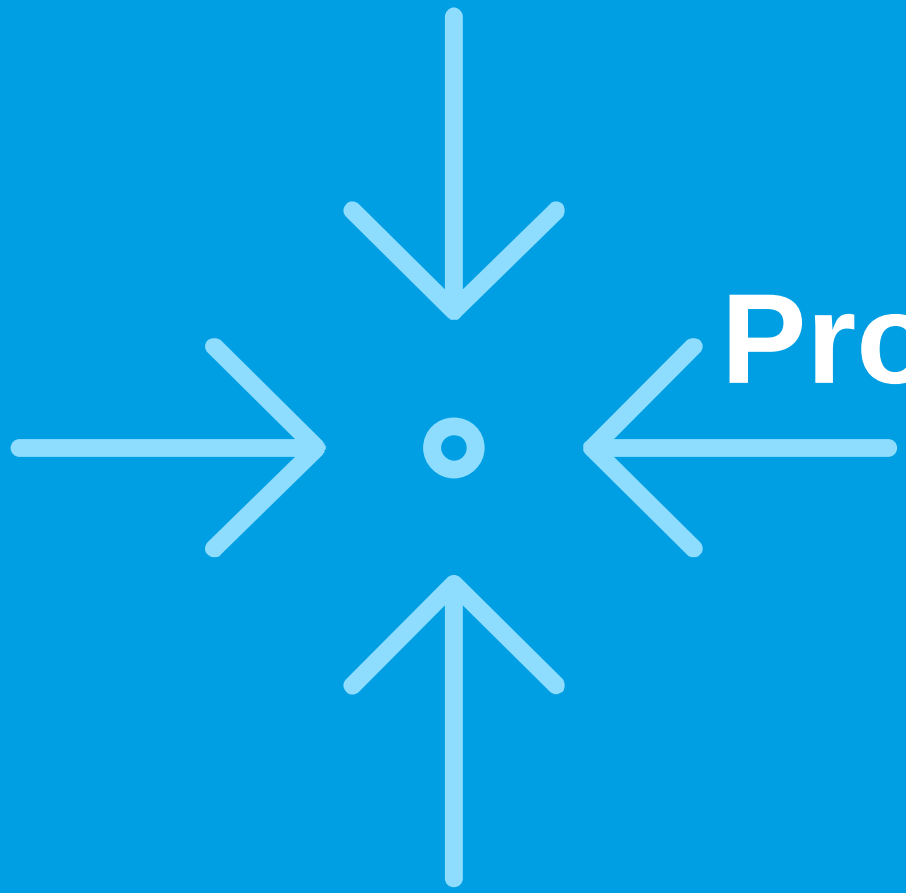
71 dependencies in pom.xml

…submodules inherit dependencies from parent

+90 dependencies, **68 redundant**

+85 dependencies **= 150** for many components

itestra
be excellent



Kapitel 03

Program analysis

Even before parsing…
Challenge 1: What to analyse?

itestra
be excellent

Overview of files of the system			
Element	Count	Size	LoC
pck	3.020	30.745.055	456.948
pkb	2.778	142.439.755	2.628.494
prc	2.535	15.274.161	297.631
sql	2.200	55.685.710	755.727
xsql	1.776	50.483.562	402.813
tab	1.041	2.211.270	63.755
con	892	607.939	22.522
ind	676	366.396	13.820
gnt	628	584.826	14.967
alt	586	780.065	18.100
fmb	541	196.747.264	2.624.875
vw	497	1.306.854	32.962
sqs	394	146.528	6.143
typ	372	669.962	18.508
PRC	111	1.368.738	32.268
gif	106	27.083	317
SQL	100	449.848	13.141
PCK	87	1.009.317	21.719
PKB	76	4.343.572	99.530
fnc	73	266.855	7.496
ctl	64	96.930	2.183
ins	44	1.014.262	19.695
trg	44	328.703	7.356
pll	39	5.038.080	55.856
seq	32	11.957	592
upd	24	38.045	1.122
idx	22	10.834	442
bas	15	76.254	2.444
sh	15	16.550	533
VW	14	37.808	1.105
frm	11	177.086	5.510
ico	11	10.970	13
TAB	10	10.888	314

INS	8	118.006	2.564
spb	7	526.600	12.607
com	7	33.122	698
ALT	7	11.880	347
csv	6	5.846	464
IND	6	1.652	72
mmb	5	3.538.944	13.587
dat	4	777.169	19.688
java	4	41.362	1.004
txt	4	118.436	250
bat	4	2.816	69
CON	4	1.629	58
lib	4	3.208	34
zip	3	23.312.763	173.204
XSQL	3	2.060.955	9.963
pc	3	121.168	3.034
olb	3	335.872	994
spc	3	30.322	649
TRG	3	18.875	547
PLS	3	8.524	310
vbp	3	4.457	147
SPB	2	222.555	5.367
jar	2	278.599	2.012
tpb	2	27.262	527
pbd	2	18.115	379
lst	2	4.871	374
dll	2	139.264	351
xsqll	2	30.312	304
h	2	8.024	239
SPC	2	7.839	168
cre	2	5.900	133
FNC	2	4.338	131
avt	2	3.208	88
lin	2	996	28
frx	2	3.532	27

def	2	483	20
vbw	2	592	14
xls	1	171.008	4.124
rdf	1	188.416	2.898
cls	1	19.938	577
pls	1	20.328	486
fct	1	25.763	440
c	1	4.429	139
exe	1	49.152	106
UPD	1	2.016	97
syn	1	5.616	87
Pkb	1	2.246	[illegible]
make	1	1.683	[illegible]
TYP	1	2.049	[illegible]
del	1	1.213	[illegible]
pla	1	769	[illegible]
gnt2	1	908	29
PKS	1	787	21
CTL	1	619	19
bmp	1	11.078	19
sps	1	656	19
DEP	1	265	16
grn	1	516	15
dis	1	738	13
pl	1	353	12
hst	1	1.811	10
lus	1	141	7
SCC	1	198	5
exp	1	1.061	3
lnk	1	1.474	3

Upper/lower case!

Which modules are relevant?

Even before parsing…
Challenge 1: What to analyse?

#WRK1416.txt
A020.txt
A024.txt
A027.txt
A029.txt
A030.txt
A031.txt
A031OT.txt
A032.txt
A035.txt
A038.txt
A039.txt
A040.txt
A041.txt
A042.txt
A043.txt
A044.txt
A046.txt
A047.txt
A048.txt
A049.txt
A050.txt
14.799 Elemente

A whole application landscape in one folder…

ABPA.txt
ABP
ABS
ABS
ABU
ABU
ABV
ABV
ABV
ABV
ABV
ABV
ABV
ABZ
ABZ
ADA
ADA
ADABTO.txt
ADABUCH.txt

Encoding problems

ZV909.txt
ZV910.txt
ZV950.txt
ZV951.txt
ZWDDEL.txt
ZWK1.txt
ZWK2.txt
ZWKA.txt
ZWKAH01M.txt
ZWKAH02M.txt
ZWKAH03M.txt
ZWKAH04M.txt
ZWKAH05M.txt
ZWKAH06M.txt
ZWKAH07M.txt
ZWKAH08M.txt
ZWKAH09M.txt
ZWKAH10M.txt
ZWKAM03.txt
ZWKAM04.txt

C

Assembler

COBOL

Compile-Log

Is it complete?
Does it compile?

Challenge 1: What to analyse? „Version Control“-Systems

Task: get a version of the source 1 year ago

Many, but not all projects do have a version control system

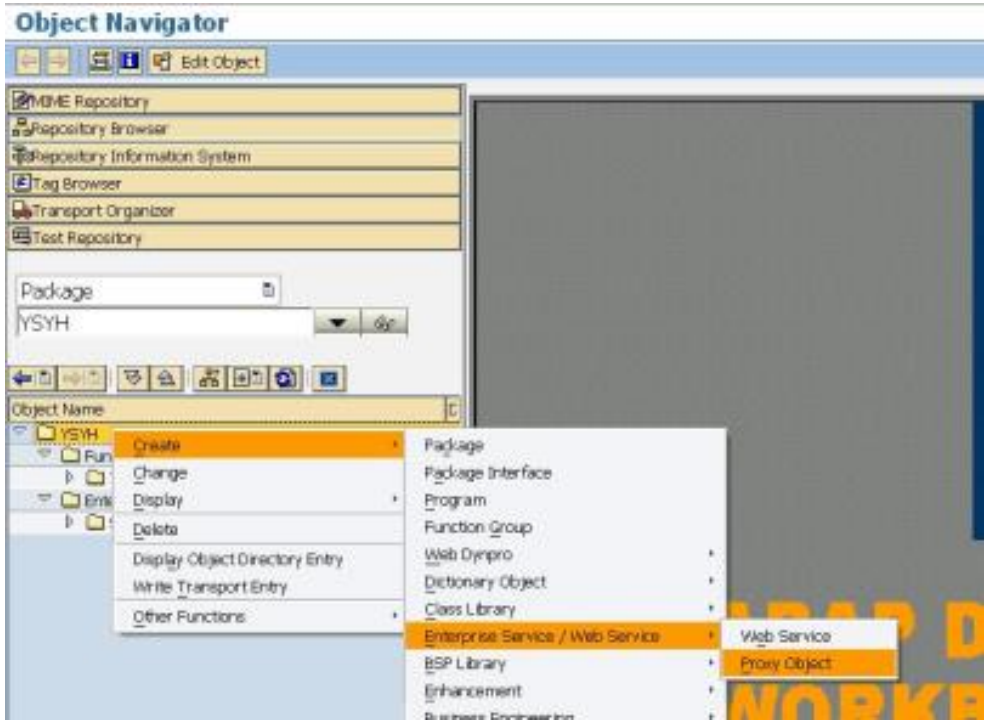
E.g. MDA models

Some version control system treat each file separately and do not support an overall view / version number

CVS w/o tags

Many VCS on mainframe

SAP

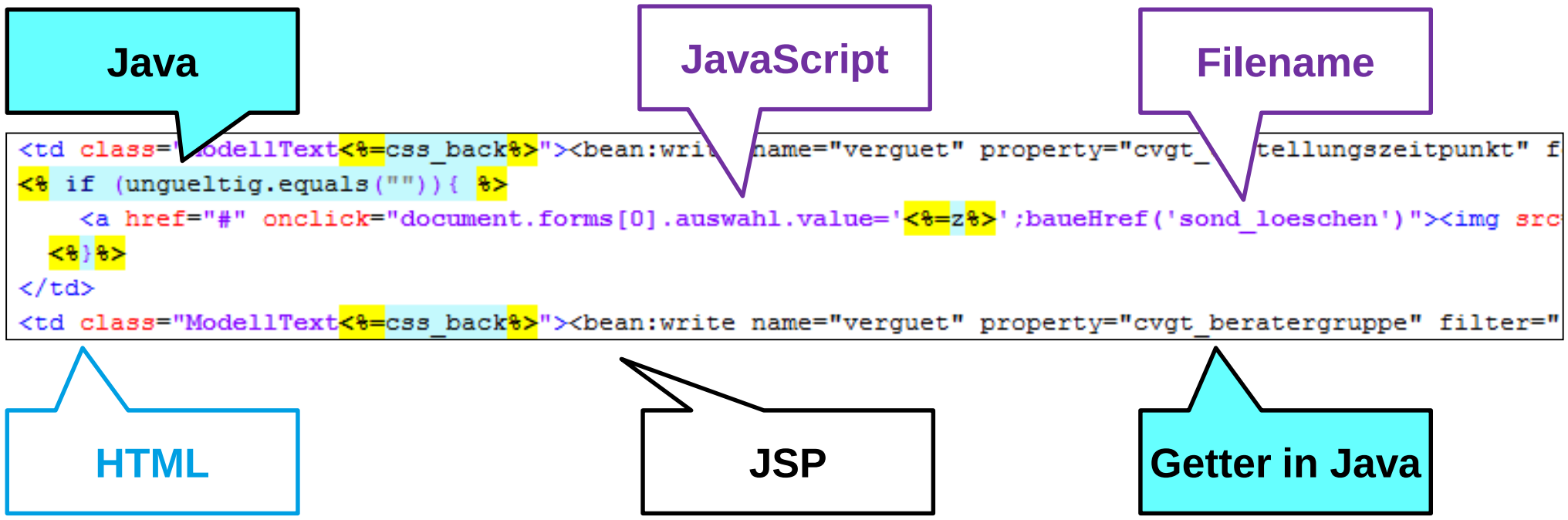


Challenge 2: Syntax
Partially generated modules

```
/* Generated - edit only in designated sections */
public class VertragHandler {
        private VertragHandler instance;
        public void handle(Event e) {
/* --- BEGIN MANUAL SECTION --- */
                int id = e.getSelectedID();
                Vertrag v = vertragDAO.load(id);
                …
/* --- END MANUAL SECTION --- */
        }
…
```

Goal: ignore the generated sections for analysis
Cutting the manual section is possible –
but what remains is not valid Java!

Challenge 2: Syntax Language mixture

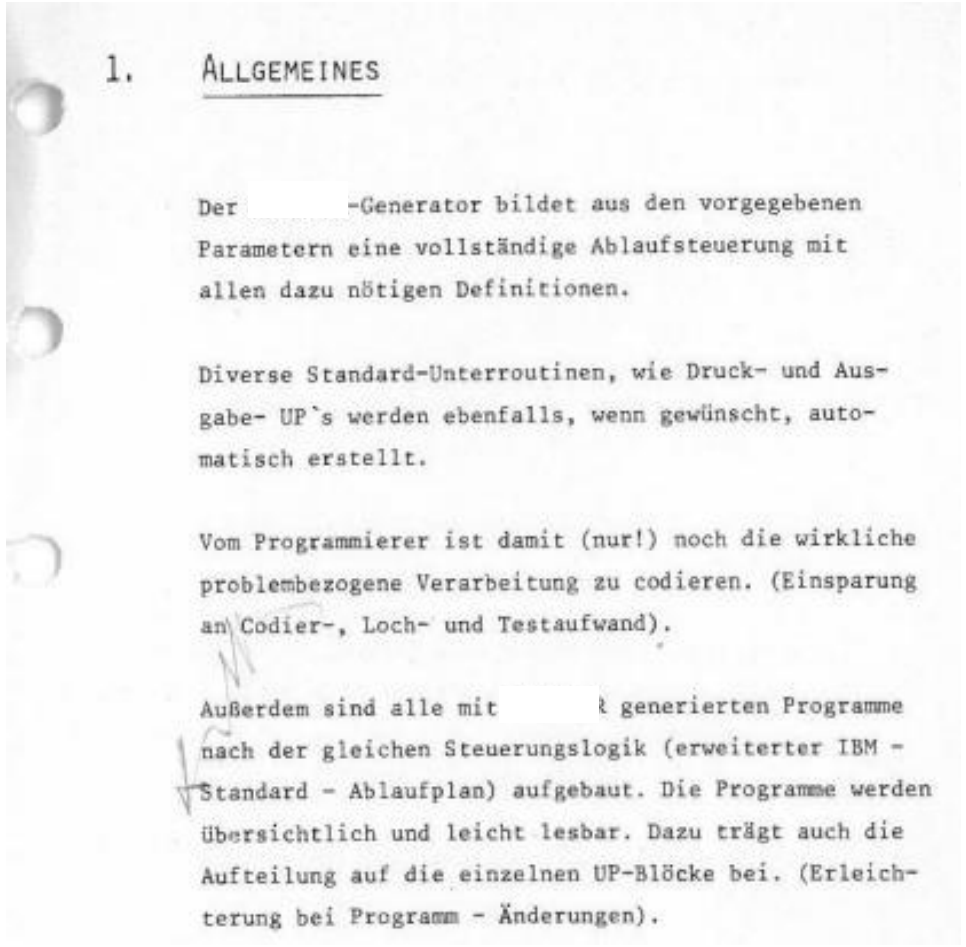


Challenge 2: Syntax
Undigestable comments

itestra
be excellent

```
1  ================================================================================
2  == ÜBERWACHUNG DES JOBS ZUR ANTRAGSSCHNELLERFASSUNG
3  == Achtung: Dieses ist kein ausführbares Skript, sondern enthält EINZELDE
4  ==          Statements zur selektiven Ausführung im Navigator
5  ================================================================================
6    > ABFRAGE DER JOBS
7  select * from dba_jobs where schema_user =
```

Challenge 2: Syntax
Preprocessors or other homegrown tools



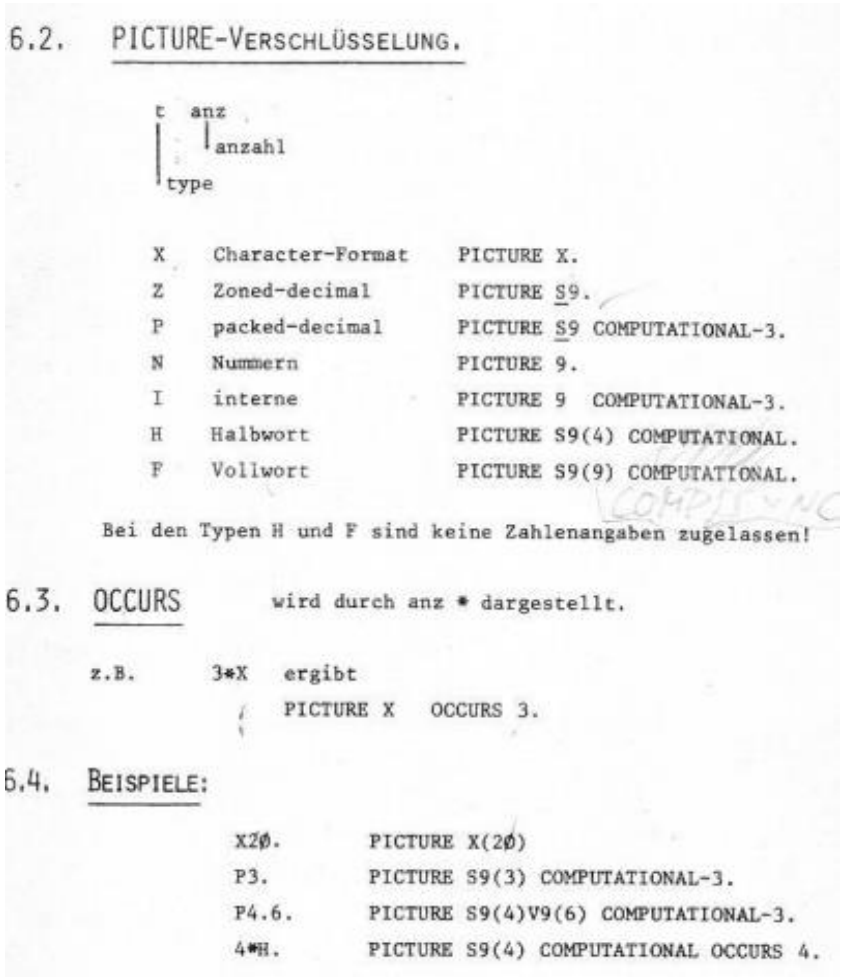
1. ALLGEMEINES

Der -Generator bildet aus den vorgegebenen Parametern eine vollständige Ablaufsteuerung mit allen dazu nötigen Definitionen.

Diverse Standard-Unterroutinen, wie Druck- und Ausgabe- UP`s werden ebenfalls, wenn gewünscht, automatisch erstellt.

Vom Programmierer ist damit (nur!) noch die wirkliche problembezogene Verarbeitung zu codieren. (Einsparung an Codier-, Loch- und Testaufwand).

Außerdem sind alle mit k generierten Programme nach der gleichen Steuerungslogik (erweiterter IBM - Standard - Ablaufplan) aufgebaut. Die Programme werden übersichtlich und leicht lesbar. Dazu trägt auch die Aufteilung auf die einzelnen UP-Blöcke bei. (Erleichterung bei Programm - Änderungen).



6.2. PICTURE-VERSCHLÜSSELUNG.

```
t  anz
|   |anzahl
|type
```

X	Character-Format	PICTURE X.
Z	Zoned-decimal	PICTURE S9.
P	packed-decimal	PICTURE S9 COMPUTATIONAL-3.
N	Nummern	PICTURE 9.
I	interne	PICTURE 9 COMPUTATIONAL-3.
H	Halbwort	PICTURE S9(4) COMPUTATIONAL.
F	Vollwort	PICTURE S9(9) COMPUTATIONAL.

Bei den Typen H und F sind keine Zahlenangaben zugelassen!

6.3. OCCURS wird durch anz * dargestellt.

z.B. 3*X ergibt
PICTURE X OCCURS 3.

6.4. BEISPIELE:

X2Ø.	PICTURE X(2Ø)
P3.	PICTURE S9(3) COMPUTATIONAL-3.
P4.6.	PICTURE S9(4)V9(6) COMPUTATIONAL-3.
4*H.	PICTURE S9(4) COMPUTATIONAL OCCURS 4.

Challenge 2: Syntax Exotic Grammars

itestra
be excellent

Keywords allowed as identifiers

Valid PL/I: **`IF IF = THEN THEN IF = ELSE ELSE IF = THEN END`**

Non-LL/LR parseable (?)

Valid COBOL: **IF A = B OR C = 1 OR 2 OR 3**

Column-based:

```
Ï*
C           PIAMOD    IFEQ 'MOD'                          B004
C           #LASTP    ANDNE'VO1004'
C                     MOVELPIACAL    #PCAL   10 P
C                     MOVELPIAPGM    #PPGM   10 P
C                     MOVEL#LASTP    #PLAS   10 P
C                     MOVEL'VO1004'  PIACAL
C                     ITER
C                     ENDIF                               E004
Î*
C                     ENDIF                               E003
Î*
C           #LASTP    IFEQ 'VO1004'                       B003
C                     MOVEL#PLAS     #LASTP
C                     MOVEL#PPGM     PIAPGM
Î*
C           PINDAB    IFEQ *ZEROS                         B004
C           PIAMO7    ANDEQ'1'
```

Challenge 2: Syntax Exotic Grammars

itestra
be excellent

Keywords allowed as identifiers

Valid PL/I: `IF IF = THEN THEN IF = ELSE ELSE IF = THEN END`

May be a function or variable

Non-LL/LR parseable (?)

Valid COBOL: **IF A = B OR C = 1 OR 2 OR 3**

Column-based:

```
Î*
C           PIAMOD    IFEQ 'MOD'                          B004
C           #LASTP    ANDNE'V01004'
C                     MOVELPIACAL     #PCAL   10 P
C                     MOVELPIAPGM     #PPGM   10 P
C                     MOVEL#LASTP     #PLAS   10 P
C                     MOVEL'V01004'   PIACAL
C                     ITER
C                     ENDIF                               E004
Î*
C                     ENDIF                               E003
Î*
C           #LASTP    IFEQ 'V01004'                       B003
C                     MOVEL#PLAS      #LASTP
C                     MOVEL#PPGM      PIAPGM
Î*
C           PINDAB    IFEQ *ZEROS                         B004
C           PIAM07    ANDEQ'1'
```

Challenge 2: Syntax Exotic Grammars

itestra
be excellent

Keywords allowed as identifiers

Valid PL/I: `IF IF = THEN THEN IF = ELSE ELSE IF = THEN END`

May be a function or variable

Non-LL/LR parseable (?)

Valid COBOL: `IF A = B OR C = 1 OR 2 OR 3`

Column-based:

```
Î*
C           PIAMOD    IFEQ 'MOD'                          B004
C           #LASTP    ANDNE'V01004'
C                     MOVELPIACAL     #PCAL   10 P
C                     MOVELPIAPGM     #PPGM   10 P
C                     MOVEL#LASTP     #PLAS   10 P
C                     MOVEL'V01004'   PIACAL
C                     ITER
C                     ENDIF                               E004
Î*
C                     ENDIF                               E003
Î*
C           #LASTP    IFEQ 'V01004'                       B003
C                     MOVEL#PLAS      #LASTP
C                     MOVEL#PPGM      PIAPGM
Î*
C           PINDAB    IFEQ *ZEROS                         B004
C           PIAM07    ANDEQ'1'
```

Challenge 2: Syntax Exotic Grammars

Keywords allowed as identifiers

Valid PL/I: `IF IF = THEN THEN IF = ELSE ELSE IF = THEN END`

May be a function or variable

Non-LL/LR parseable (?)

Valid COBOL: `IF A = B OR C = 1 OR 2 OR 3`

Column-based:

```
Î*
C           PIAMOD    IFEQ 'MOD'                        B004
C           #LASTP    ANDNE'VO1004'
C                     MOVELPIACAL     #PCAL   10 P
C                     MOVELPIAPGM     #PPGM   10 P
C                     MOVEL#LASTP     #PLAS   10 P
C                     MOVEL'VO1004'   PIACAL
C                     ITER
C                     ENDIF                             E004
Î*
C                     ENDIF                             E003
Î*
C           #LASTP    IFEQ 'VO1004'                     B003
C                     MOVEL#PLAS      #LASTP
C                     MOVEL#PPGM      PIAPGM
Î*
C           PINDAB    IFEQ *ZEROS                       B004
C           PIAMO7    ANDEQ'1'
```

Challenge 3: Special Cases

itestra
be excellent

Needed for detecting Dead Code:

External extry points

Calls / references

Challenge 3: Special Cases

itestra
be excellent

Needed for detecting Dead Code:

External extry points

Calls / references

…but there are a number of special cases, e.g.

Reflection and dynamic calls (worst case: name is dynamically concatenated)

Declarative objekt creation: e.g. EJB class is never referenced explicitely!

References in configuration files

```
<assembly-descriptor>
   <!-- Default interceptor that will apply to all methods for all beans in deployment -->
   <interceptor-binding>
      <ejb-name>*</ejb-name>
      <interceptor-class>org.jboss.tutorial.interceptor.bean.DefaultInterceptor</interceptor-class>
      <interceptor-class>org.jboss.tutorial.interceptor.bean.AnotherInterceptor</interceptor-class>
   </interceptor-binding>
   ...
```

A real-world analysis **must** consider these!
Just asking for ideal prerequisites does‘nt work!

Approaches

itestra
be excellent

Be lazy & stand on the shoulders of giants:

Use automisation where appropriate

Use existing tools/ building blocks if fitting

But know the limits:

Use Brains resp. manual steps for special cases and context specifica

Configure, extend or build tools to automate steps that can be automated but exceed the capabilities of existing tools

80/20

Combination of manual and automated techniques

```
c:\Temp\demo>preprocessing --set-ext-from-content=interactive -d . -o ..\demo2
* INFO  : Converting charset from windows-1252 (windows-1252) to UTF-8 (UTF-8).
* INFO  : Processing $ARCCOM1
* INFO  : Processing $ARCDCL1
* INFO  : Processing $ARCREQ1
* INFO  : Processing $ARCST11
* WARN  : Couldn't guess language - no matches.
 /** %INCLUDE ARCST1   ***ANFANG**********************************************/00000010
 /*-------------------------------------------------------------------------*/00000020
 /*    PROJEKT         :                                                    */00000030
 /*    PROGRAMMIERER   :                                                    */00000040
 /*    VERSION         :                                                    */00000050
 /*    ERSTELLT        :                                                    */00000060
 /*-------------------------------------------------------------------------*/00000070
 /*****************                                 *********************/00000080
 /*                                                                         */00000090
 /*  VORLAGE FUER                                                           */00000100
...
  END ;                                                                    00000330
                                                                           00000340
 H_STATUS3              = ' ';                                            00000350
                                                                           00000360
 /** %INCLUDE ARCST1   ***ENDE  **********************************************/00000370
Please type desired extension (without dot) or just press Enter for unknown:
```

Tools & Building blocks

itestra
be excellent

Existing **universal tools**, e.g.:

ShellScript & Unix Tools, Python

Excel (!)

FileLocator, Astrogrep etc.

Building blocks for new challenges:

Lexer / Lexer generator for different languages

Parser, Parse Utilities, Parser generator

Dependency graphs

Preprocessing tools

Eclipse

Keep your skripts – you will run them again!

Example: Document creation scripts

Customer with 1000s of IBM ASF skripts

Proprietary skripting language
for document generation

```
[STRASSENZEILE,&STRASSE. &HAUSNUMMER.]
```

High infrastructure cost, no future

```
.if &L'&$TXTIDKOST. > 0
```

Migration needed

```
.se lauf = &lauf. + 1
```

```
[INSTITUTSBLZ,&$BLZ.]
```

Questions from preparatory analysis e.g.

How is a value calculated?

Which fields depend (directly) on a parameter?

Which parameters are assignes/passed but never used?

Approach: create parser using ANTLR, create output similar to JavaDoc

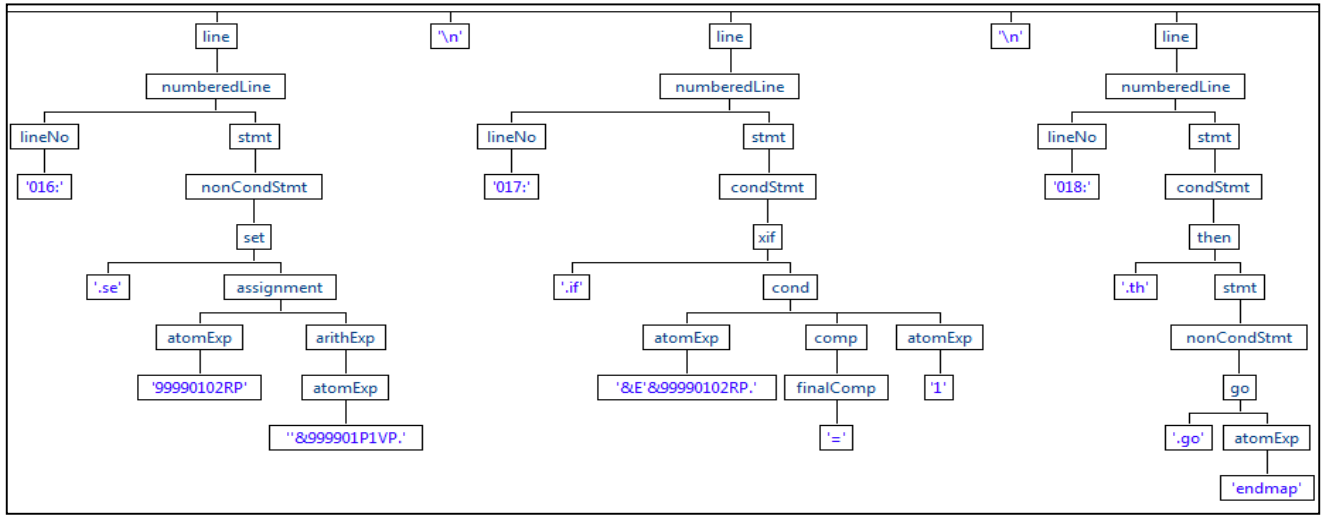
Example ctd.: Tooling

Grammar for lexing/parsing ASF script

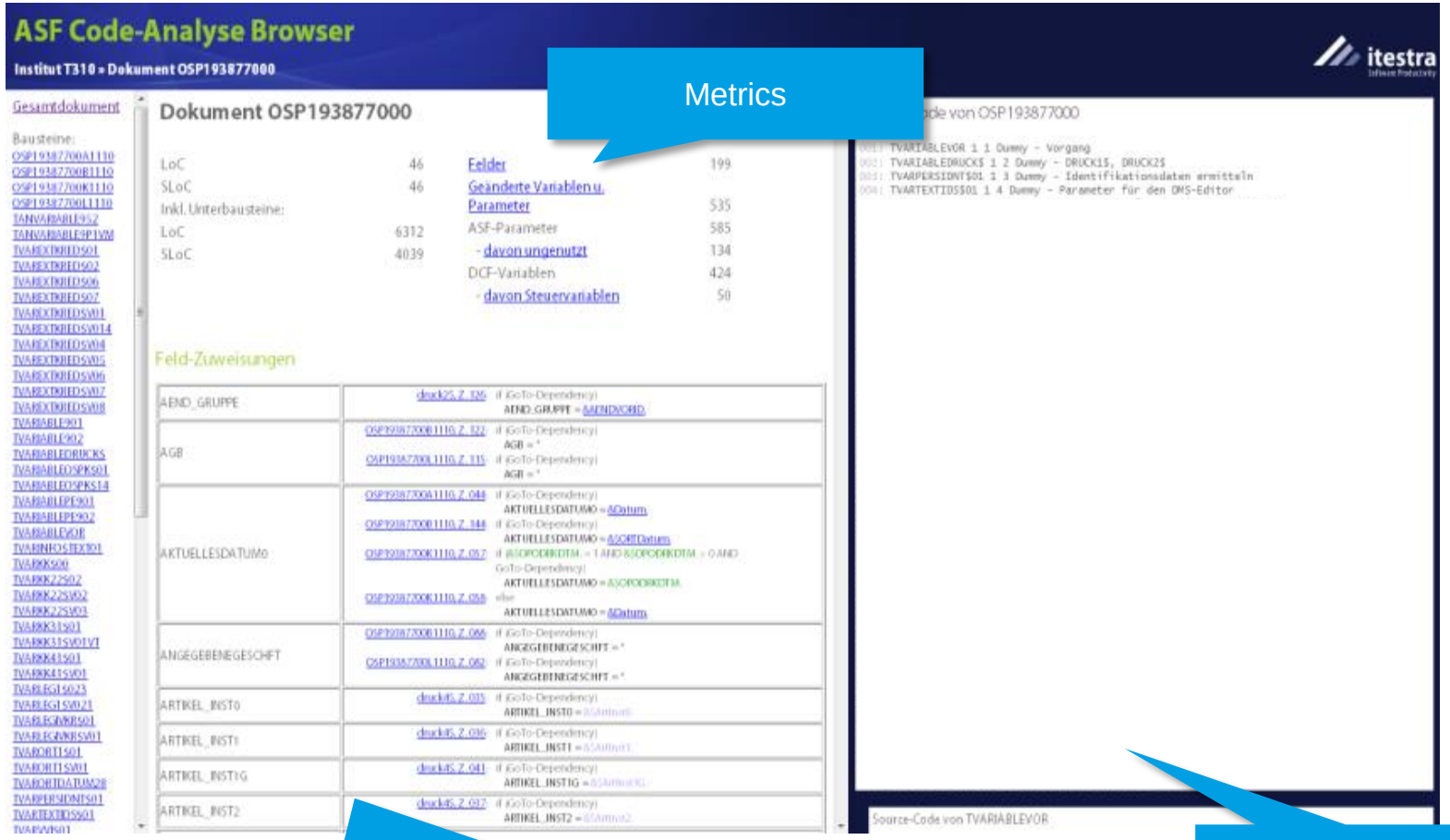
```
infile  :   line (NEWLINE line)* NEWLINE? EOF;

line    :   (docName
            |   bsBegin
            |   imEnd
            |   numberedLine
            );

docName :   ID;

bsBegin :   '======' ID '======';

imEnd   :   '------' ID '------';


numberedLine
    :       lineNo (
                comment
                |   bsInclude
                |   stmt (';' stmt)* (';' comment)?
            );

stmt    :   condStmt | nonCondStmt;

condStmt
    :       (xif
            |   then
            |   xelse
            |   and
            |   or
            );

nonCondStmt
    :       (set
            |   include
            |   go
            |   goMark
            |   killNull
            |   ef
            |   rechnen
            |   rechne
```

(partially shown)

…creates syntax-tree:



Example ctd.: Resulting output



Metrics

Assignments including link to Code Line, conditions etc.

Original source

Example: Parsing conditionals

Application landscape containing **hard-coded references and conditions** on products and tariffs (> 10.000 in 4 Mio SLoC)

Goal: **1. Find them**
2. If possible, **replace them** by call to rule engine

Approach: Custom-built condition normalization, built upon an existing parser (~ 10 days effort)

```
*
DECIDE ON FIRST VALUE OF SN3071A.E-BAUSPARTARIF
  VALUE 6
     IF SN3071A.E-TARIFVARIANTE = 'C' OR = 'D'
       TD3071A.A-VARIANTE  := SN3071A.E-TARIFVARIANTE
     ELSE
       TD3071A.A-VARIANTE  := '0' /* Bei allen T3 ist 0 als Variante ein-
     END-IF                       /* getragen, auáer C/D
```

and(BAUSPARTARIF=6, **or(**TARIFVARIANTE='C', TARIFVARIANTE='D'**))**

```
IF BSV-BESTAND.BAUSPARTARIF = 2 OR = 9
   #GRUND-VERTRAG-IGNORIEREN := #C-VERTRAG-RIESTER
   ESCAPE TOP
END-IF
```

```
IF IBS.BAUSPARTARIF = 9 OR = 2
  G2410-ABSNAME(1)              := 'A2039'

-ABSNAME(1)                     := 'A2034'
```

Both equivalent to

or(BAUSPARTARIF=2, BAUSPARTARIF=9**)**

Heuristics

itestra
be excellent

Depending on the Use Case, **fuzzy results** may be acceptable and allow the processing of large data sets

often **not possible within a compiler**, e.g. in code generation …

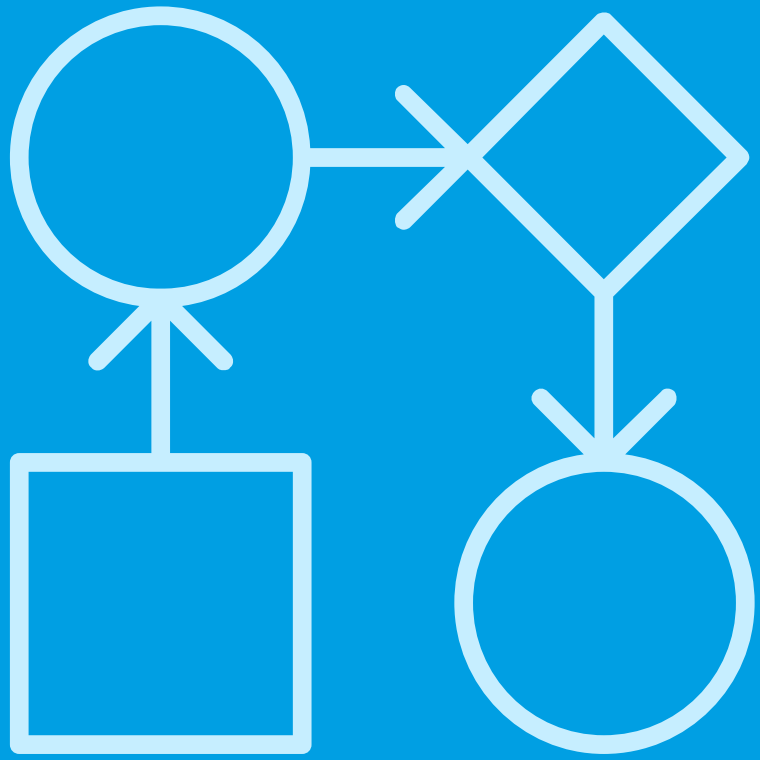
…but **acceptable for code metrics**

1. **Ignore** unknow stuff (maybe issue a warning)

2. Analyses based on **tokens** instead of **full parsing**
   (or use Shallow Parsing / Island Parsing)
   ⇨ Robustness in case of unexpected content
   ⇨ Speed & Memory consumption

Example: COBOL unused include file analysis requires only

- detection of include files (no occurrence of **`PROCEDURE DIVISION`**)
- detection of include statements (**`COPY <filename>`**)



Kapitel 04

Programm transformation

Automated migration

Tempting idea: **automated legacy-systems migration**

COBOL → Java / C#

ADABAS, IMS, VSAM → DB2

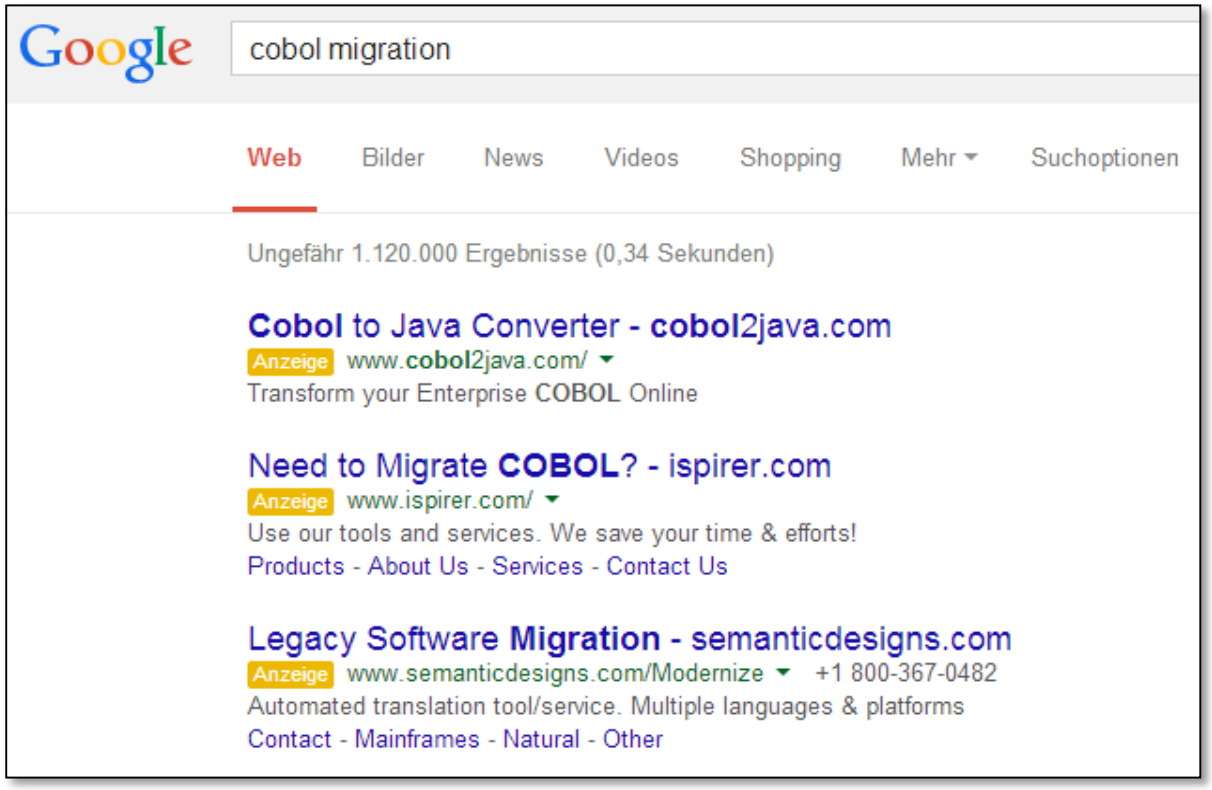
etc.

Claims:

Cheap

Fast

Error-free

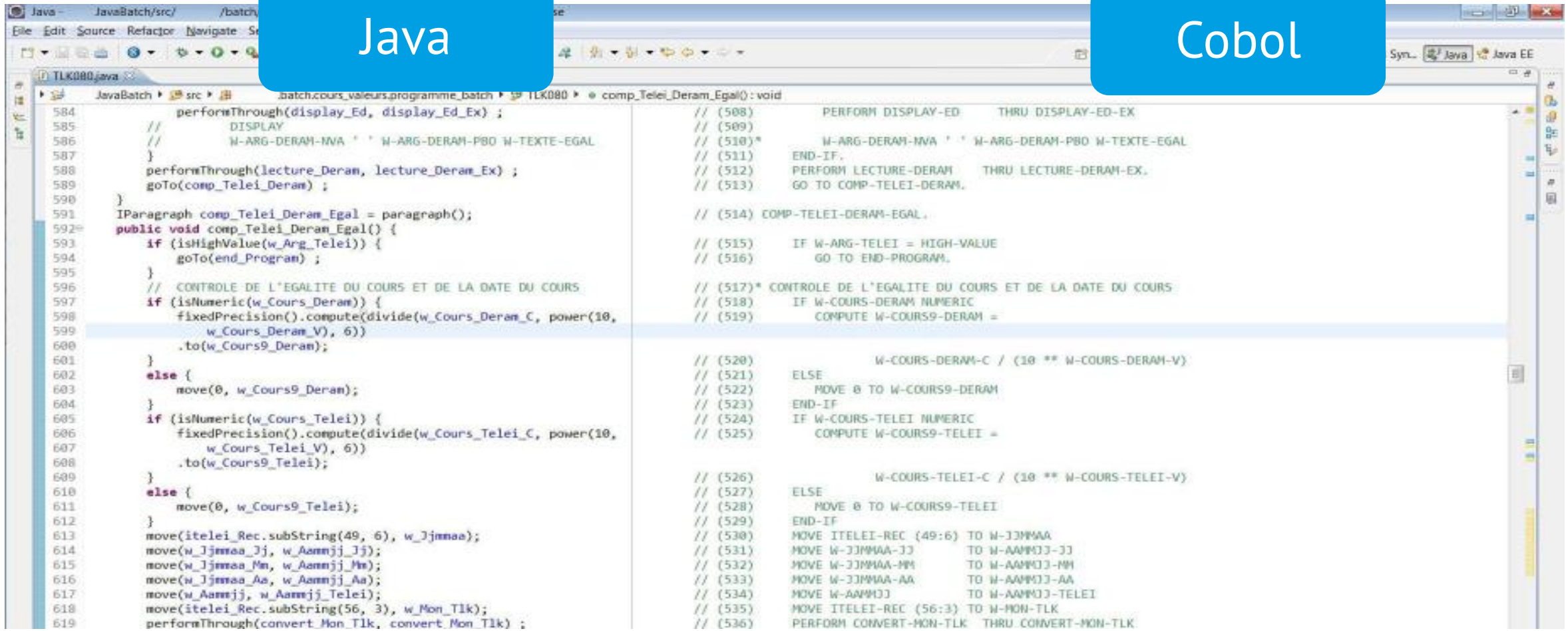


Translation examples: NACA

itestra
be excellent

Java

Cobol



```
584             performThrough(display_Ed, display_Ed_Ex) ;
585         //          DISPLAY
586         //          W-ARG-DERAM-NVA ' ' W-ARG-DERAM-PBO W-TEXTE-EGAL
587         }
588         performThrough(lecture_Deram, lecture_Deram_Ex) ;
589         goTo(comp_Telei_Deram) ;
590     }
591     IParagraph comp_Telei_Deram_Egal = paragraph();
592     public void comp_Telei_Deram_Egal() {
593         if (isHighValue(w_Arg_Telei)) {
594             goTo(end_Program) ;
595         }
596         //  CONTROLE DE L'EGALITE DU COURS ET DE LA DATE DU COURS
597         if (isNumeric(w_Cours_Deram)) {
598             fixedPrecision().compute(divide(w_Cours_Deram_C, power(10,
599                 w_Cours_Deram_V), 6))
600             .to(w_Cours9_Deram);
601         }
602         else {
603             move(0, w_Cours9_Deram);
604         }
605         if (isNumeric(w_Cours_Telei)) {
606             fixedPrecision().compute(divide(w_Cours_Telei_C, power(10,
607                 w_Cours_Telei_V), 6))
608             .to(w_Cours9_Telei);
609         }
610         else {
611             move(0, w_Cours9_Telei);
612         }
613         move(itelei_Rec.subString(49, 6), w_Jjmmaa);
614         move(w_Jjmmaa_Jj, w_Aammjj_Jj);
615         move(w_Jjmmaa_Mm, w_Aammjj_Mm);
616         move(w_Jjmmaa_Aa, w_Aammjj_Aa);
617         move(w_Aammjj, w_Aammjj_Telei);
618         move(itelei_Rec.subString(56, 3), w_Mon_Tlk);
619         performThrough(convert_Mon_Tlk, convert_Mon_Tlk) ;
```

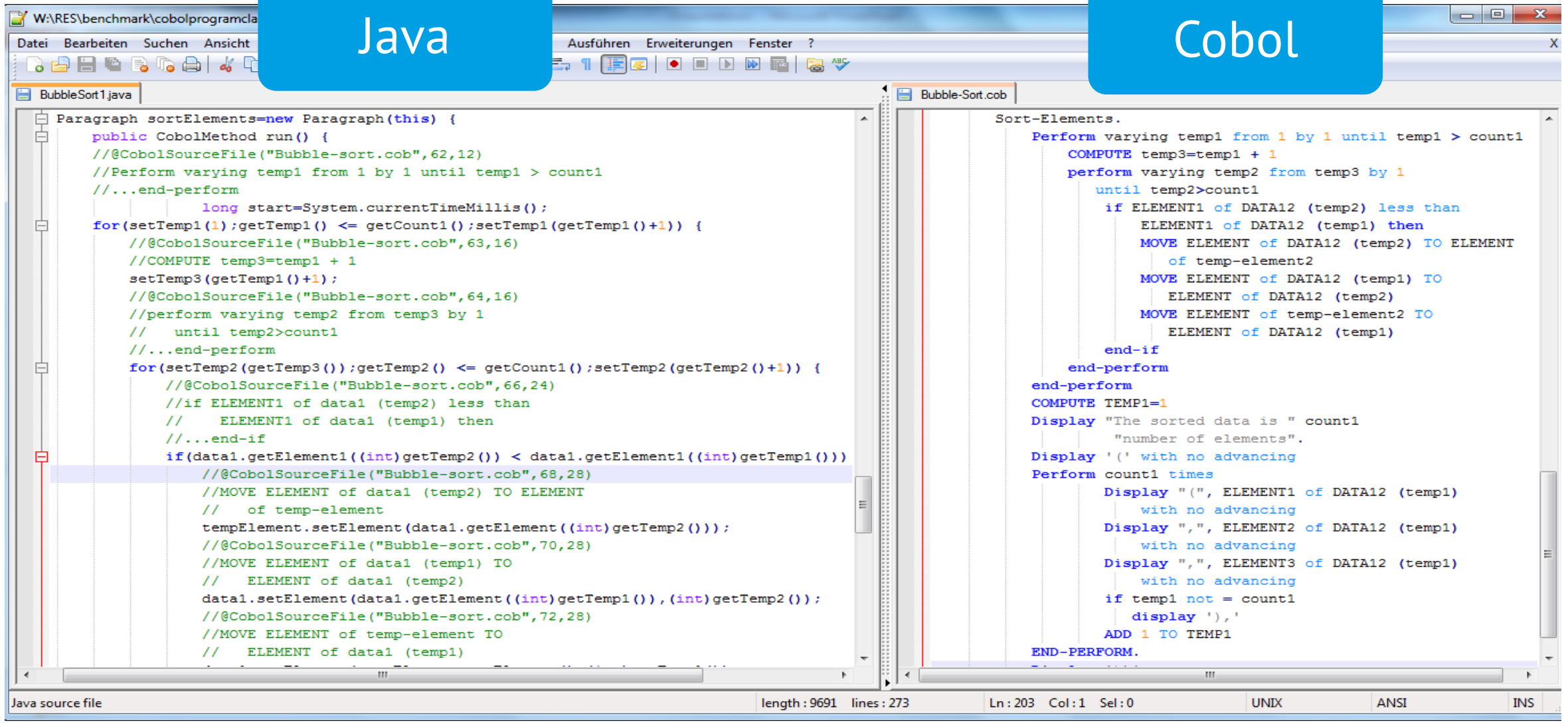
```
// (508)          PERFORM DISPLAY-ED      THRU DISPLAY-ED-EX
// (509)
// (510)*         W-ARG-DERAM-NVA ' ' W-ARG-DERAM-PBO W-TEXTE-EGAL
// (511)      END-IF.
// (512)      PERFORM LECTURE-DERAM    THRU LECTURE-DERAM-EX.
// (513)      GO TO COMP-TELEI-DERAM.

// (514) COMP-TELEI-DERAM-EGAL.

// (515)      IF W-ARG-TELEI = HIGH-VALUE
// (516)         GO TO END-PROGRAM.

// (517)* CONTROLE DE L'EGALITE DU COURS ET DE LA DATE DU COURS
// (518)      IF W-COURS-DERAM NUMERIC
// (519)         COMPUTE W-COURS9-DERAM =

// (520)                W-COURS-DERAM-C / (10 ** W-COURS-DERAM-V)
// (521)      ELSE
// (522)         MOVE 0 TO W-COURS9-DERAM
// (523)      END-IF
// (524)      IF W-COURS-TELEI NUMERIC
// (525)         COMPUTE W-COURS9-TELEI =

// (526)                W-COURS-TELEI-C / (10 ** W-COURS-TELEI-V)
// (527)      ELSE
// (528)         MOVE 0 TO W-COURS9-TELEI
// (529)      END-IF
// (530)      MOVE ITELEI-REC (49:6) TO W-JJMMAA
// (531)      MOVE W-JJMMAA-JJ       TO W-AAMMJJ-JJ
// (532)      MOVE W-JJMMAA-MM       TO W-AAMMJJ-MM
// (533)      MOVE W-JJMMAA-AA       TO W-AAMMJJ-AA
// (534)      MOVE W-AAMMJJ          TO W-AAMMJJ-TELEI
// (535)      MOVE ITELEI-REC (56:3) TO W-MON-TLK
// (536)      PERFORM CONVERT-MON-TLK  THRU CONVERT-MON-TLK
```

Translation examples: OpenCobol2Java

itestra
be excellent

Java

Cobol



Structure, Redundancy avoidance

itestra
be excellent

```
WHEN ('Ä')
  DO;
     ZW_FD37_O(I3) = 'A';
     I3 = I3 + 1;
     ZW_FD37_O(I3) = 'E';
  END;
WHEN ('Ö')
  DO;
     ZW_FD37_O(I3) = 'O';
     I3 = I3 + 1;
     ZW_FD37_O(I3) = 'E';
  END;
WHEN ('Ü')
  DO;
     ZW_FD37_O(I3) = 'U';
     I3 = I3 + 1;
     ZW_FD37_O(I3) = 'E';
  END;
WHEN ('ß')
  DO;
     ZW_FD37_O(I3) = 'S';
     I3 = I3 + 1;
     ZW_FD37_O(I3) = 'S';
  END;
```

```
StringUtils.replaceUmlaut(fd37);
```

…what about

- adequate **naming**
- **service-orientation**
- **processing strategies**, e.g. batch pipeline
- **datamodel** normalization

?

Evaluation

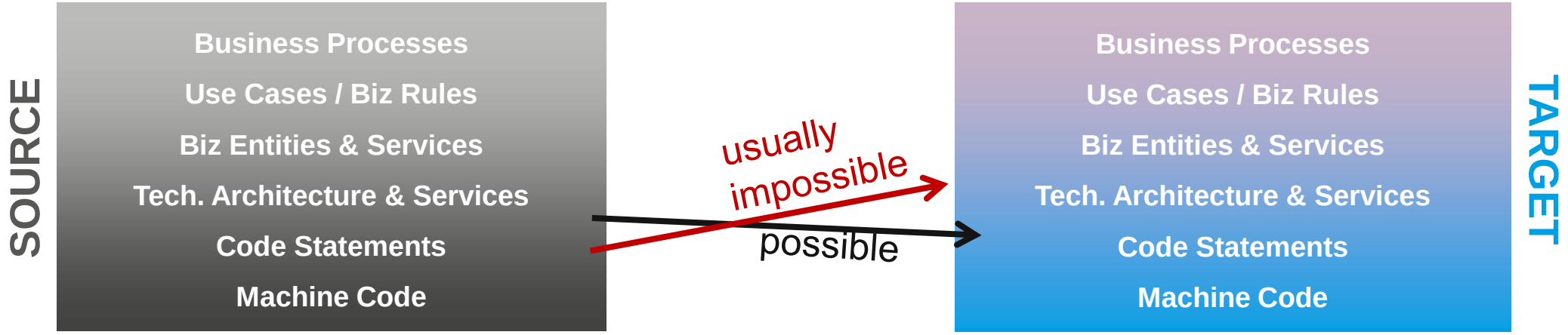
itestra
be excellent

Mistakes and problems of the legacy language are **conserved**

The **amount of code** usually remains the same or **even increases**

Usually, migrated code has to be post-processed manually to obtain a (realistically) human-readable form

⇨ **Maintenance** will be **at least as expensive** as before or even costlier
(and **performance** usually **decreases** too!)



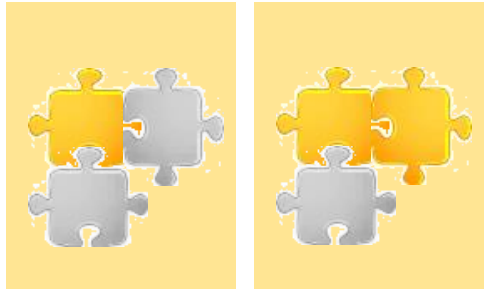
Evaluation

Nevertheless, there are Use Cases for automated migration

- (fast) **reduction of infrastructure cost**, i.e. CPU or license, through platform change
- **Incremental migration** scenario

Situation:
Old code on
legacy platform

Automated migration:
Run old code
on new platform



Incremental replacement
of old components by
newly designed ones
with possible interaction
of old and new parts

Target:
new code
on new platform

- **Partial code generation** for interface classes, data migration etc.

Example: Interface generation

Incremental migration of COBOL system to C#

Data migration, test result comparison and interface between old&new requires access to “flat” in old format (dozens of formats, 100s of fields)

⇨ Generator creates C# access classes from COBOL copy

```
      03    BA-AA-TAB-TBGR  OCCURS 99 INDEXED BY IBAAAJ IBA
        04  BA-AA-BGUV-NRX.
        05  BA-AA-BGUV-NR   PIC X(2).
        05  BA-AA-BGUV-NR3  PIC X.
*                           BG SCHLUESSEL
*                           MEHRFACHKENNUNG
        04  BA-AA-BGUV-MINR PIC X(20).
*                           BG MITGLIEDSNUMMER
        04  BA-AA-BGUV-GTSX.
        05  BA-AA-BGUV-GTS-NR PIC X(8).
        05  BA-AA-BGUV-GTS-BG PIC X(2).
*                           GTS GEFAHRTARIFSTELLE STANDARD
*                           GTS BG SCHLUESSEL
        04  BA-AA-BGUV-KZ   PIC X.
*                           1 = BG STANDARDANNAHME
*                           2 = BG INDIVIDUELL
        04  BA-AA-BGUV-GR   PIC X(3).
*                           ABGABEGRUND
        04  BA-AA-BGUV-PA   PIC X.
*                             = FREI
*                           1 = IN PERSONALSTAMM VERWENDET
        04  BA-AA-BGUV-REST PIC X(47).
```

```
using Datenformate.Attributes;
using Kern.Types;
using System;

namespace Exportformat
{

// Data access class generated by StructView
[SatzFesteBreite(10478)]
public class B1CBAAAX {
    // at offset 0 / 0x0
..

    // at offset 1440 / 0x5a0
    [FesteBreite(1441, 2)]
    public string _BA_AA_BGUV_NR_0 { get { return BA_AA_BGUV_NR[0];}
    set { BA_AA_BGUV_NR[0] = value; } }

    [FesteBreite(1526, 2)]
    public string _BA_AA_BGUV_NR_1 { get { return BA_AA_BGUV_NR[1];}
    set { BA_AA_BGUV_NR[1] = value; } }
```

Example: Reengineering

Situation

Legal insurance system (> 1,4 Mio. Verträge)

Legacy technology (700.000 LOC RPG, AS400),
only 2 developers with Know How

Barely any innovation, changes expensive

Threat: Loss of market share

Solution: Iterative modernisation (Java, Portal)

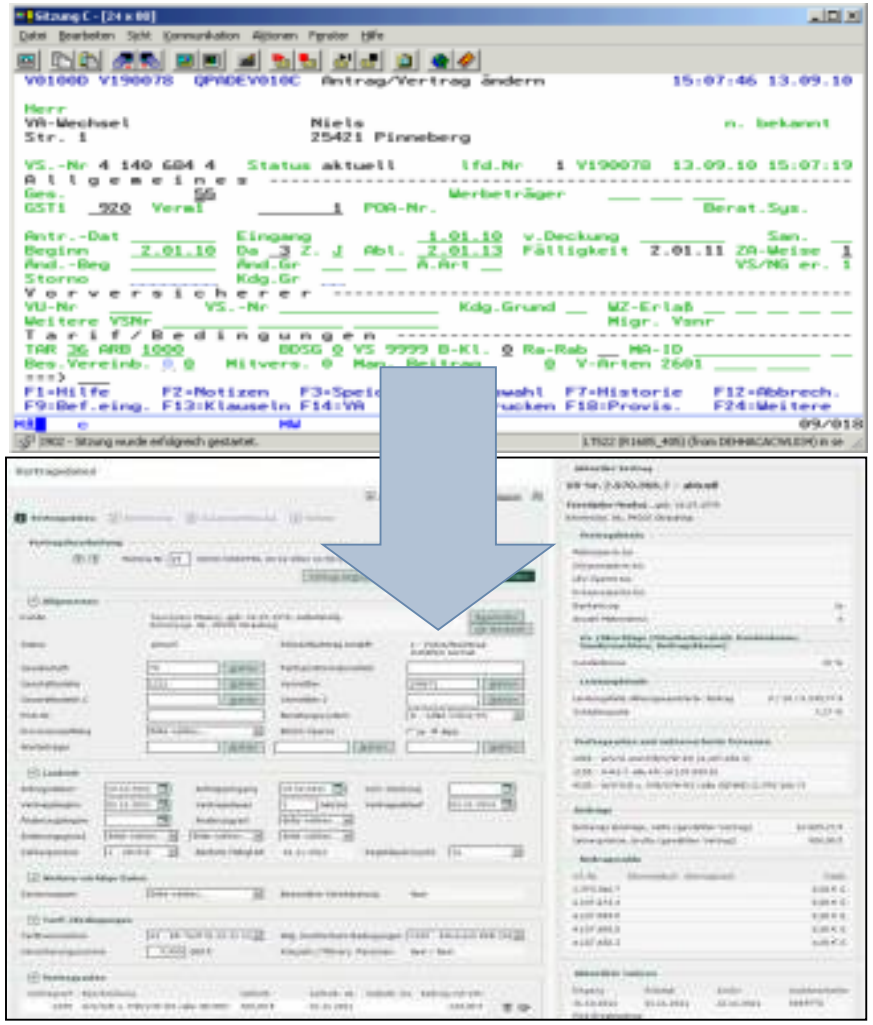
Reverse Engineering of business logic

Minimal support by business experts needed

Incremental migration of business functionality

Redocumentation

⇨ **mostly manual work**
…but necessary for reasonable results!



Kapitel 6

Weiteres Interesse?

Informatiker gefragt!

- Festanstellung
- Werkstudententätigkeit
- Praktikum
- Abschlussarbeiten (Bachelor & Master)

Bewerben Sie sich!
jobs@itestra.de

Praxis-Workshop
RWTH Aachen, 09.01.2019

In kleinen Teams entwickelt ihr anhand eines Praxisbeispiels selbstständig die Architektur für ein Informationssystem. Außerdem lernt ihr den Arbeitsalltag eines Softwarearchitekten bei itestra kennen.

- 09. Januar 2019, 9-16 Uhr
- RWTH Aachen, Informatik-Zentrum
- Anmeldung bis 02. Januar 2019

itestra.com/workshop-aachen

Kontakt

itestra GmbH

Destouchesstraße 68, 80796 München

E-Mail: jobs@itestra.com

Tel.: +49 89 381570-113

Fax: +49 89 381570-119