Ringvorlesung Softwaremigration

Der schwierige Weg aus der IT Vergangenheit

Von Harry M. Sneed
Lehrbeauftragter

Dezember, 2019

Die Software Falle

Die Entwicklung eigener maßgeschneiderter Software ist eine tödliche Falle aus der naive Anwender nicht so leicht wieder heraus kommen.

Deshalb, soll jeder Plan eine neue Anwendung zu entwickeln von einem zweiten Plan begleitet werden, wie der Anwender gedenkt aus der Anwendung wieder heraus zu kommen. Es ist viel leichter ein IT-System zu bauen als dasselbe System später zu ersetzen.

Zahlreiche Anwender weltweit sind das Opfer Ihres technischen Ehrgeizes.

Trapped in the Tar Pit of Legacy Software

TUD

Viele Wege führen in die Legacy Falle

MIG-03

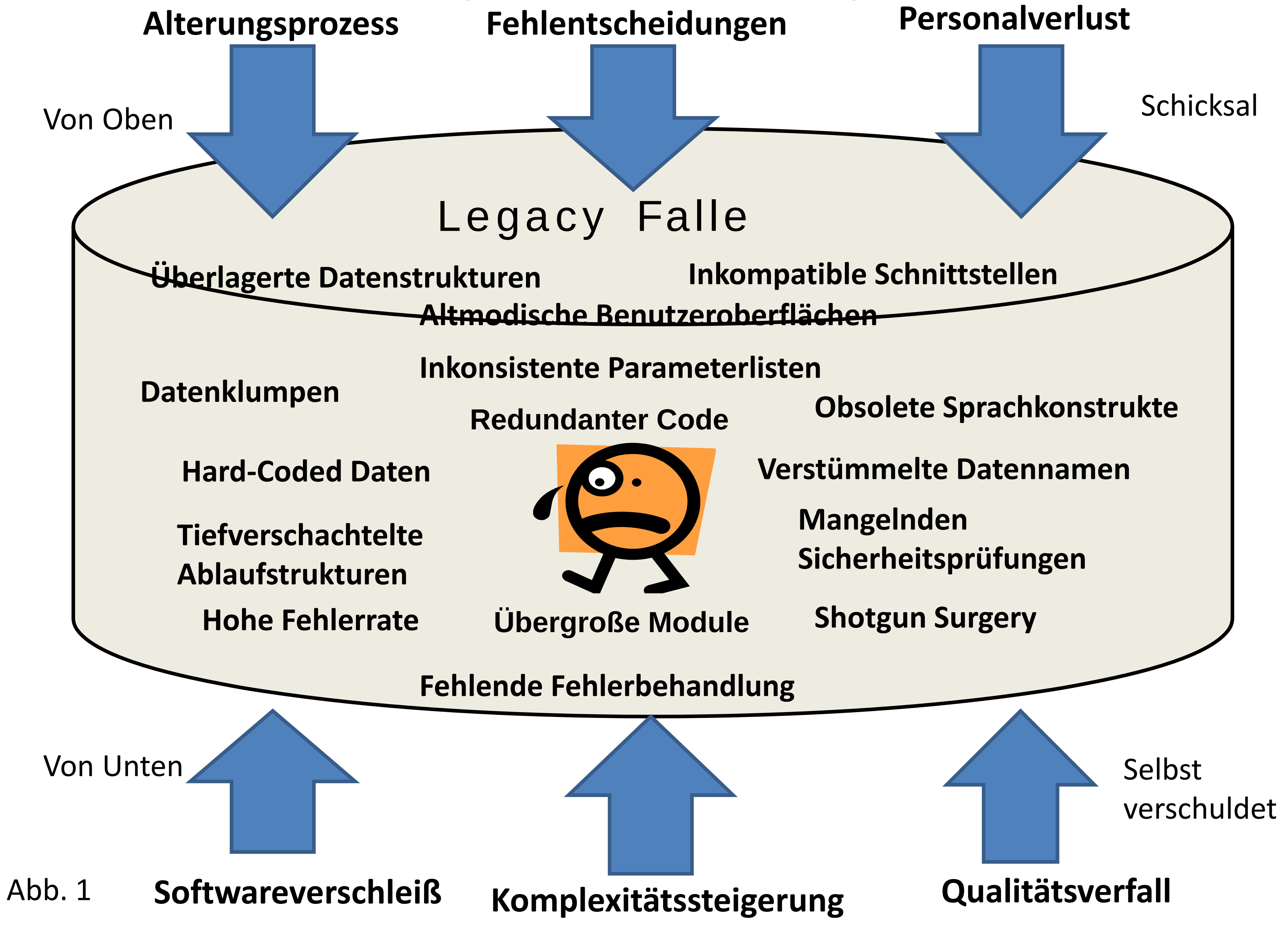


Abb. 1

Software Schulden haben zwei Ursachen

- **Unvollständigkeit**
- Unvollständiger Code
- Unvollständige Doku
- Unvollständiger Test
- Code Funktionen werden absichtlich weggelassen
- Dokumente werden nie vollendet
- Erforderliche Testfälle werden nie ausgeführt

- **Mangelhaftigkeit**
- Mangelhafter Code
- Mangelhafte Doku
- Mangelhafter Test
- Codequalität wird vorläufig geopfert
- Dokumente sind nicht konsistent und nicht normengerecht
- Testüberdeckung ist unzulänglich

Code Mängel in einer DotNet Applikation

Major deficiences sorted by number of occurences	NachbesserungsAufwand (Stunden)	
(10) Return Value is not controlled after Method Call 3435	x 1	= 3435
(22) Static Variables should be avoided in C# 3280	x 2	= 6560
(01) IO-Operations are not in a try block 2913	x 1,5	= 4370
(25) Class is not derived from a Superordinate Class 1219		
(14) Control logic exceeds maximum allowed nesting level 962	x 2	= 1924
(15) More than one Statement on a Line 856	x 0,5	= 428
(04) Data Casting should be avoided 692	x 3	= 2076
(18) Multiple Interfaces are not allowed 535	x 4	= 2140
(26) Nested Classes are not allowed 376	x 4	= 1504
(27) Returning a Function my cause an endless loop 239	x 6	= 1434
(11) Conditions should not contain an Assignment 150	x 1	= 150
(17) Try and Catch clauses do not match in last method 112	x 1	= 112
(13) Default is missing in last Switch Statement 85	x 1	= 85
(12) Case block should contain a Break statement 77	x 2	= 154
(08) Method Invocation with Array is not in a try Block 31	x 3	= 93
(07) External Variables are not allowed 29	x 2	= 58
(06) Standard IO Functions are prohibited 21	x 4	= 84
(21) Namespace should be defined in Application Classes 19	x 1	= 19
(09) There should be no global Data Definitions in C# 5	x 2	= 10
(02) Two Dimensional Arrays violate 1. Normal Form 2	x 8	= 16
		= **24.652**

Wege aus der Legacy Falle

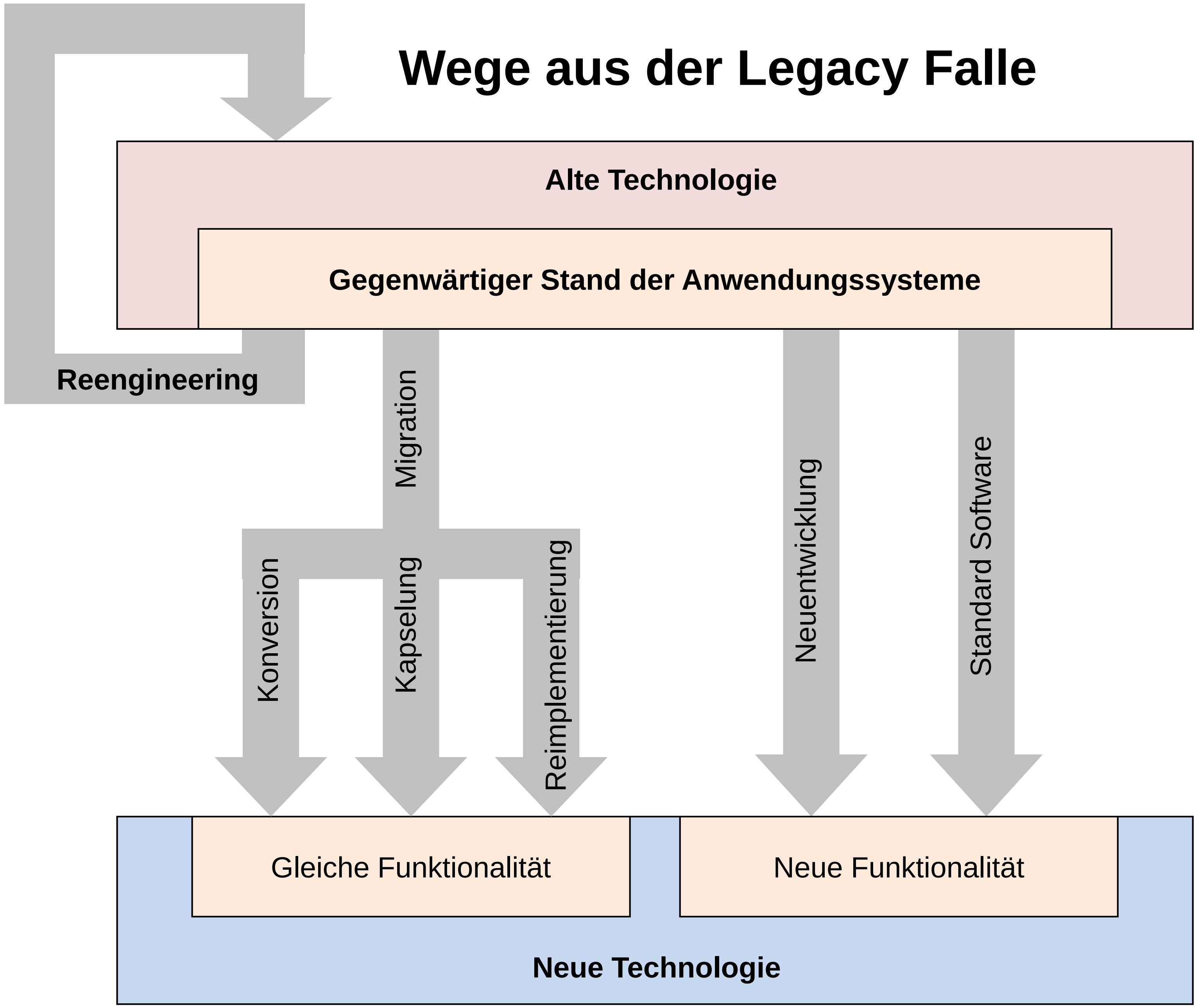


Abb. 2

1. Ausweg: Ablösung durch Standard Software

Anwender unterwerfen sich der Software Technologie

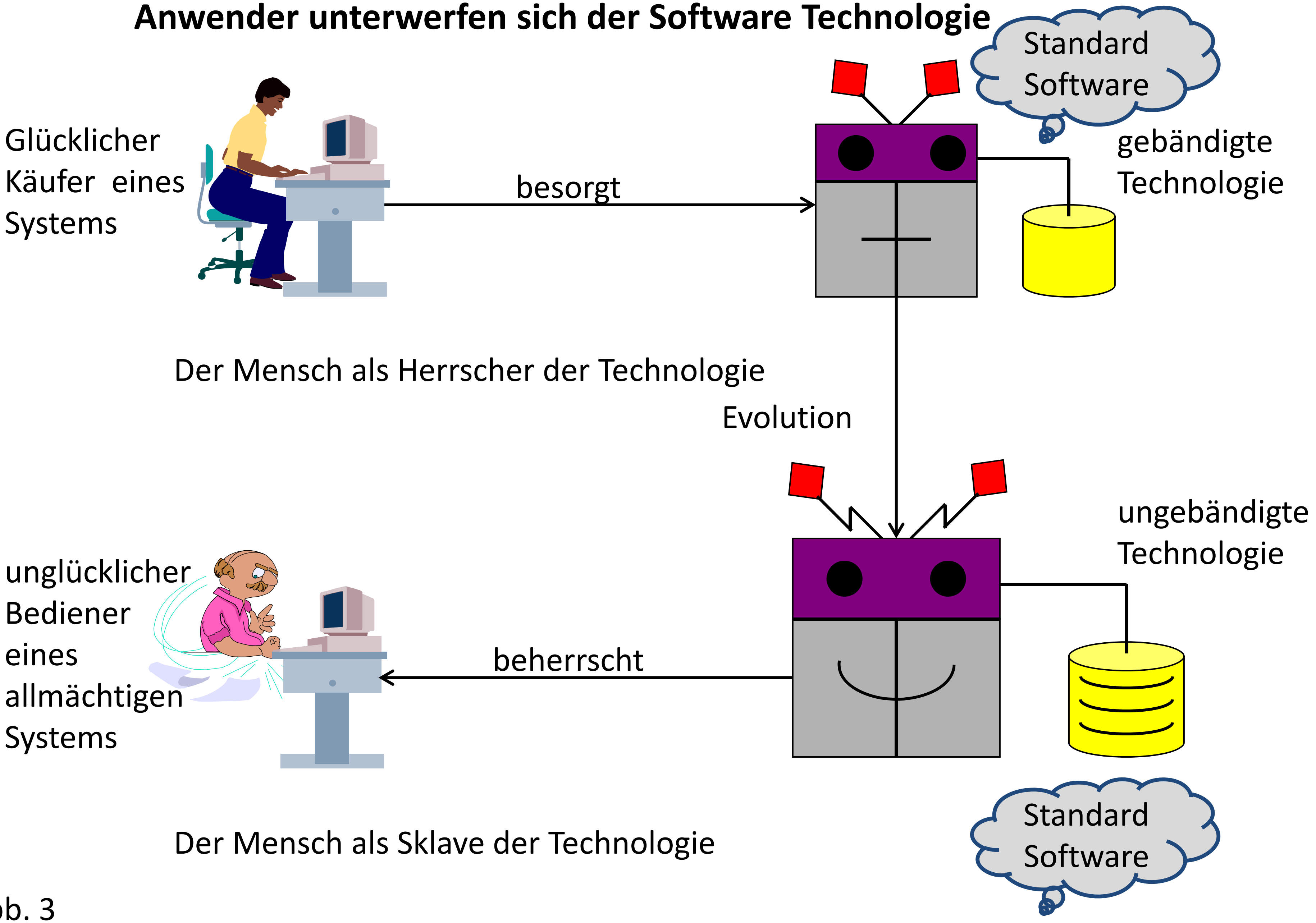

Standard Software
Glücklicher Käufer eines Systems
besorgt
gebändigte Technologie
Der Mensch als Herrscher der Technologie
Evolution
ungebändigte Technologie
unglücklicher Bediener eines allmächtigen Systems
beherrscht
Standard Software
Der Mensch als Sklave der Technologie

Abb. 3

2. Ausweg: Entwicklung eines neuen Systems neben dem Alten

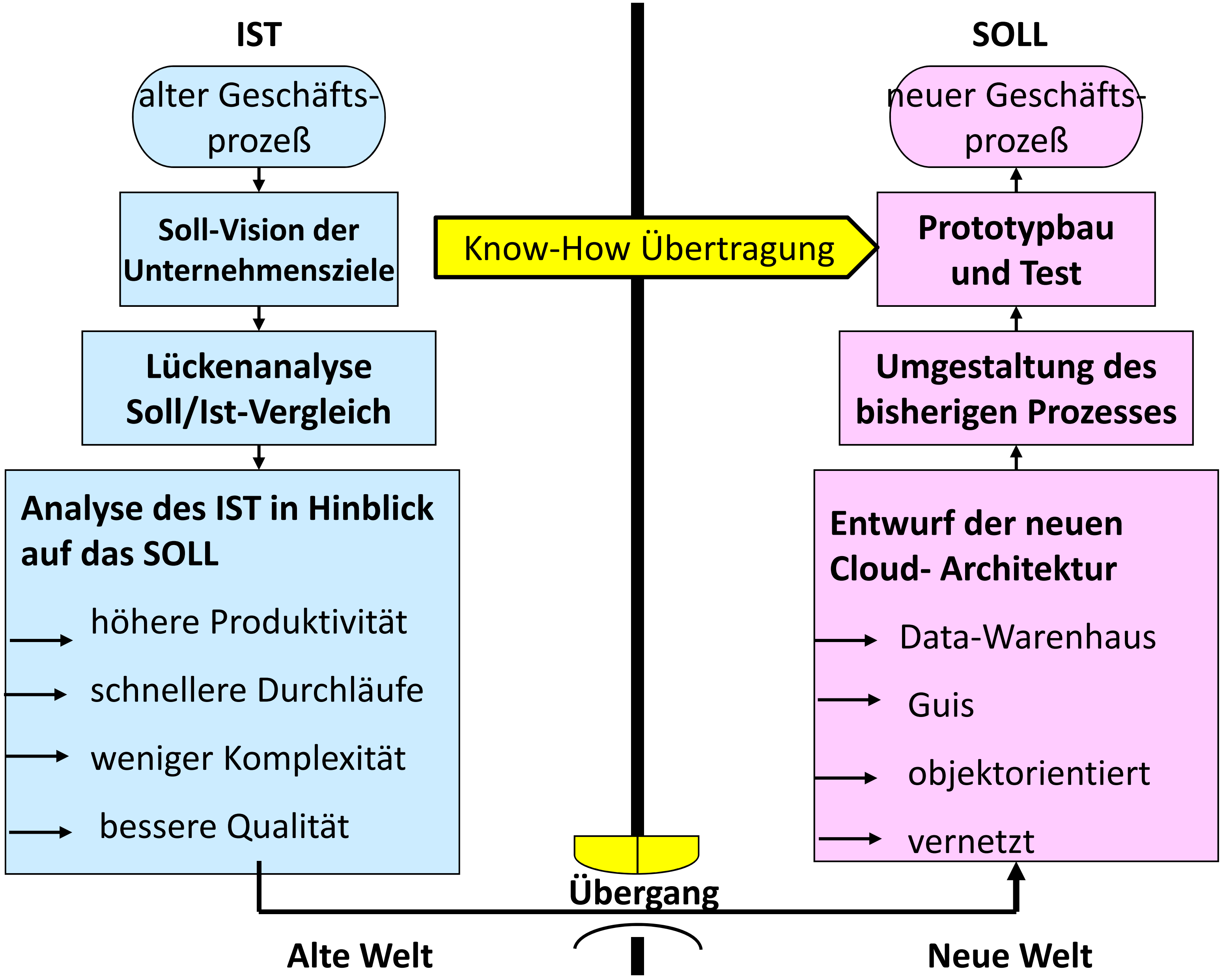


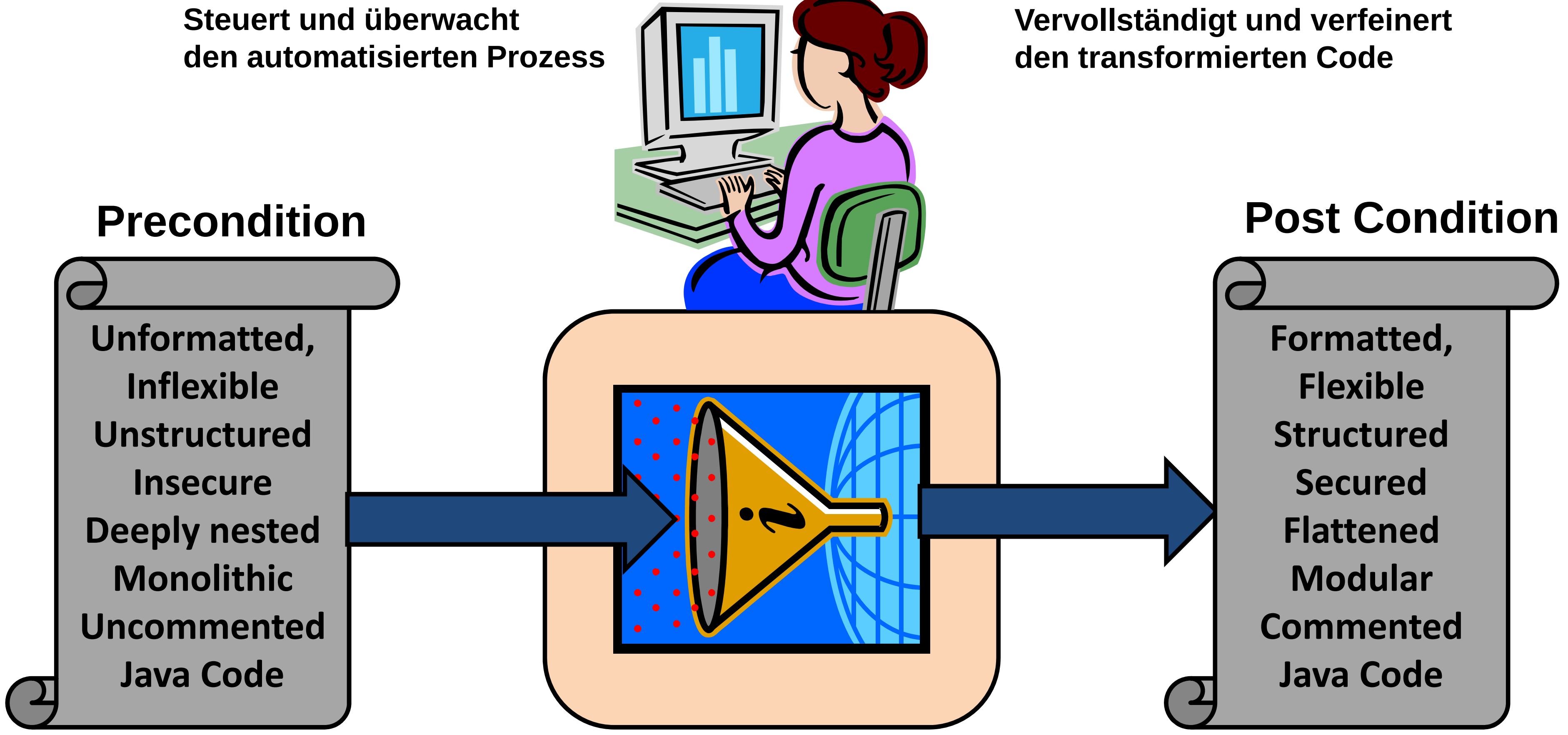
Abb. 4

3. Ausweg: Automatisierte Transformation vom alten Code

Reengineering Specialist

Steuert und überwacht den automatisierten Prozess

Vervollständigt und verfeinert den transformierten Code



Code Reengineering Process

Transformiert den Code in mehreren aufeinander folgenden Schritten

Abb. 6

Object-Oriented COBOL Recycling

by
Harry M. Sneed
SES Software-Engineering Service GmbH (1996)

In this paper a tool supported process for extracting objects from existing COBOL programs is described. The process is based on human interaction to select objects coupled with automated slicing techniques to identify all of the elementary operations which change the state of the object selected. The object is redefined within the framework of an Object-COBOL class and the elementary operations are attached to it as methods.
The result is a set of Object-COBOL classes.

1. The Rationale for Object-oriented Reengineering

Object-oriented reengineering has the goal of transforming existing procedural systems into an object-oriented architecture. The procedural systems may be structured or not, if not they should first go through structural reengineering. The result of object-oriented reengineering should, in any case, be a set of objects which perform the same functions as the previous procedures.
Object technology is definitely the predominant software trend of the 1990’s. Whether it will fulfill all of the expectations is another matter. According to the literature it should enhance maintainability, reduce the error rate, increase productivity and make the data processing world a better place to live.[1] Of course, as with all new technologies there is a lot of

marketing hype connected to it. However, for distributed applications with graphical user interfaces, there seems to be no alternative. Object orientation is a necessary precondition to realizing complex networked systems. The OMG's CORBA - Common Object Request Broker Architecture - standard is well on its way to becoming a world wide standard for accessing data and objects in a distributed computer network and for exchanging messages between objects on different computers. The key to CORBA is the IDL - Interface Definition Language – for specifying the interfaces.[2]
Through CORBA it is possible to even access legacy code on a mainframe to provide services for the clients on the periphery. This technique, known as wrapping,
is an alternative to object-oriented Reengineering.[3]

The ability to reuse code is certainly enhanced through object-oriented programming and in particular through object-oriented architectures. There is also reason to believe that the object-oriented systems are more flexible and adaptable, but the claims about improved productivity and maintainability still remain to be verified. Important is the emphasize given by the system suppliers. All the new software architectures,
be it DOE - Distributed Objects Everywhere - from SUN, Object Broker from DEC,
ORBIX from IONA or SOM - Systems Object Modelling - from IBM, are based on
object technology. Thus, in spite of the scepticism of many critics, the shift to the new paradigm is inevitable.
For better or for worse, object-orientation is the target software technology of the 90's.

The greatest obstacle to the introduction of object technology is the achievement of past technologies, i.e. the presence of so many legacy systems and the people who maintain them. As long as these systems are operating sufficiently, there is no pressing need to replace them, even if the new systems promise to be better. They would really have to be significantly better, in order to take the risk of introducing them. Wise managers have learned from past experience how difficult it is to develop new software systems. The costs of development are difficult to predict, particularly with a new technology
such as object orientation. There is also a high risk that the project will fail - according to the literature on risk assessment at least 40 %. So why should anyone in his right mind want to redevelop a working system just for the sake of being object oriented.[4]

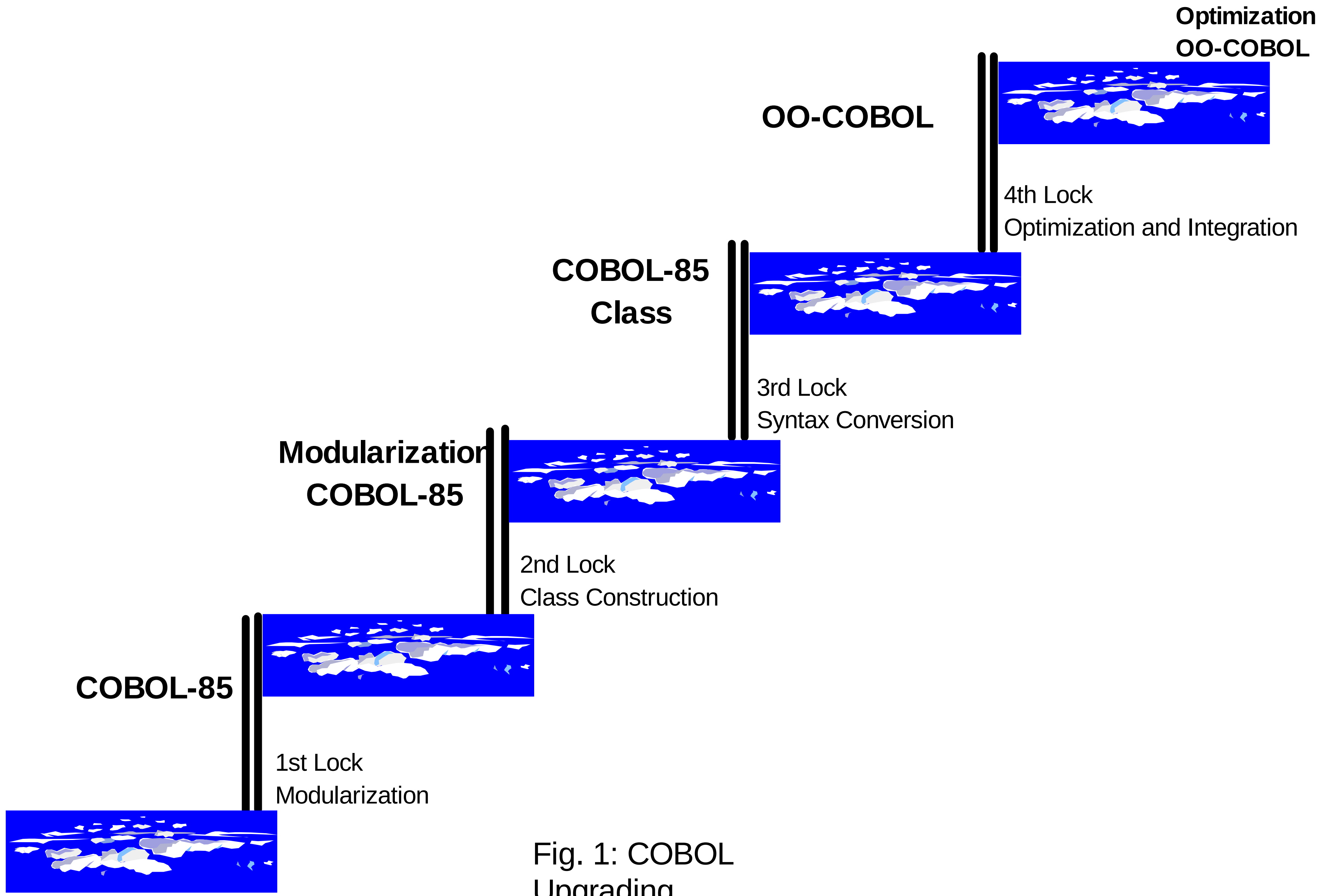


Fig. 1: COBOL Upgrading

Barriers to Code Conversion

- **Incompatible data structures**
- **Redefined data overlays**
- **Incompatible data types**
- **Non-comprehensible data names**
- **Incompatible statement types**
- **Entangled Control Flow Structures, i.e. Spahgetti Code**
- **Incompatible Data Communication & Database Frameworks like CICS and IMS**
- **Obsolete, no longer supported operating system features**

Incompatible Data Structures

In conventional procedural languages data is stored and accessed physically as fields of bytes starting at one location and ending at another. One can identify data fields by their physical starting position and their length. COBOL uses this physical storage of data fields to a great extent. Data is addressed as fields and subfields with a type and a length as shown here.

```
01 Field-A.        contains the subfields
02 Field-A1    PIC X(4).
02 Field-A2        contains the subfields
                        03 Field-A21  PIC X(8).
                        03 Field-A22  PIC X(12).
```

Each field is a continuous string of characters. With the statement

```
MOVE Field-A21 (1:4) TO Field-A22
```

the first four characters of Field-A21 are moved to the first four characters of Field-A22. This is of course impossible in an object-oriented language where each data variable is stored and retrieved separately from all the others.

In COBOL a record is a sequence of physically continuous fields. In Java a class is a set of logically related variables. To translate the MOVE statement given above the converter must create a Java string variable with the name Field_A21_1_4 to correspond to the first four characters of the COBOL field A-21.

Redefined Data Overlays

With the use of the overlay technique the incompatibility of the data becomes greater. In procedural programming languages like COBOL, PL/I and RPG it was common practice to use the same physical data storage for different logical data structures. This is referred to as overlay technique. The original idea was to save internal storage space by using the same storage space for different logical data structures. The following COBOL structure illustrates the redefined feature.

```
01 Record-A.
02 Field-A1          PIC X.    //   Value of this field indicates which overlay to use
02 Field-A2          PIC X(20).
02 Field-A21                   REDEFINES Field-A2.
03 Field-A21-1                 OCCURS 5 TIMES PIC 9(4).
                02 Field-A22                REDEFINES Field-A2.
03 Field-A22-1                 PIC  9(4).
03 Field-A22-2                 PIC X(16).
                               03 Field-A22-3      REDEFINES Field-A22-2.
04 Field-A22-31                OCCURS 4 TIMES PIC X(4).
```

To translate a redefined data structure to Java redevelopers could create two separate classes, each with it own dynamic storage. There could be a class for Field-A21 and another class for Field-A22. The class Field-A22 would contain two nested classes, one with the Fields A22-1 and A22-2 and another with the Fields A22-1 and A22-3. This solution will lead to a large number of minute classes.

Incompatible Statement Types

There are several statement types for which there are no equivalent statements in Java – for instance the string, the print and the file options. The entire file handling operations in COBOL and PL/I must be replaced by equivalent file handling operations in Java.
A typical example of a COBOL file operation for which there is no direct equivalent in Java is the reading of an indexed file by key.

```
READ DATA-FILE KEY IS DATA-NAME
INVALID KEY  MOVE 2 TO GET-RET-CODE
        NOT INVALID KEY PERFORM PROCESS-RECORD.
```

Most likely COBOL files will be converted to relational database tables and the COBOL file operations like the one above will be replaced by equivalent SQL operations:
START & READ = select, WRITE = insert, REWRITE = update, DELETE = delete
COBOL originated as a business language for preparing reports. Therefore, there is a wide selection of print commands for skipping lines, positioning on the next page, and positioning on certain columns. There are several statements of this type, statements like SEARCH and STRING.
SEARCH is a special command for searching internal tables for selected values.

```
SEARCH TABLE-X AT END GO TO FINITO,
WHEN TABLE-ELEMENT = SEARCH-KEY GO TO FOUND-IT.
```

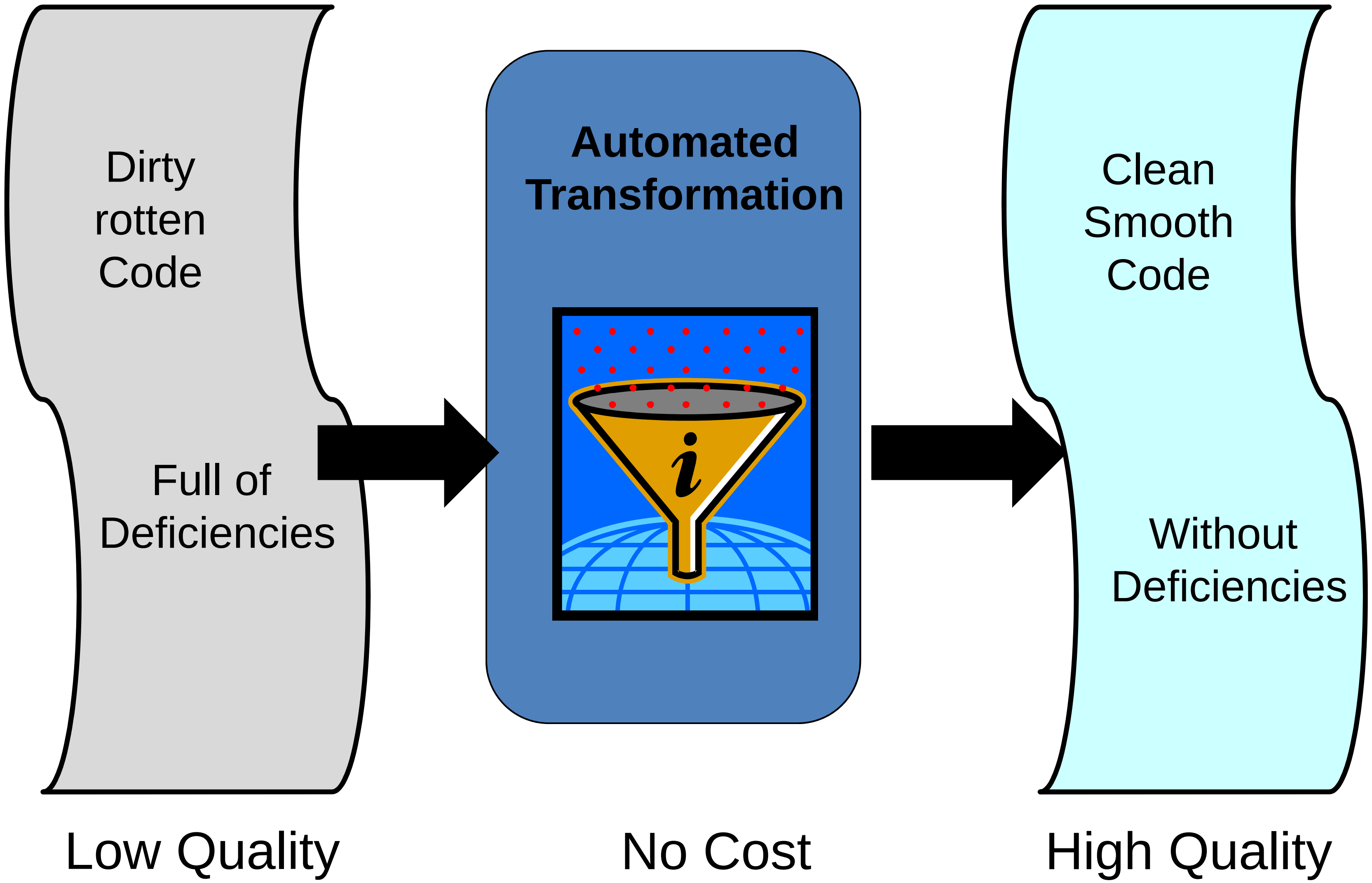
The same applies to the STRING statement which is used to concatenate string data and UNSTRING which is used to decompose substrings from string values.

```
  STRING CBL-SRC-BODY (I : 6)      DELIMITED BY SIZE
              SEQ-NR                     DELIMITED BY SIZE
             INTO COPY-NAME
```

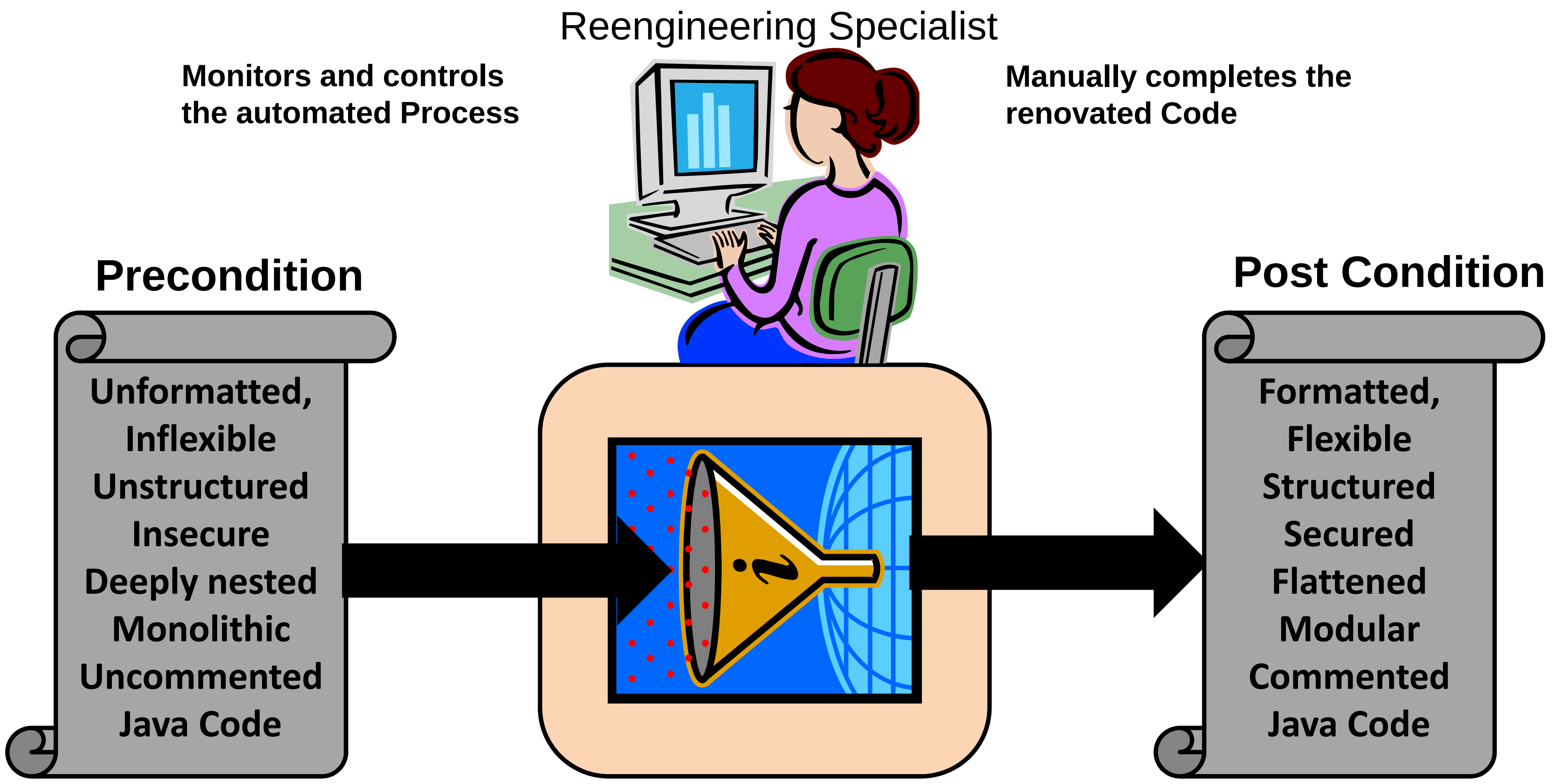
Entangled Control Flow Structures = Spahgetti Code

```
LPX1   *----------------------------------Original COBOL Code --------------------------------------
005590 0300P.
005610    IF   W01002 = "E "  GO TO 0350P.
005630    IF   W01020-25 = SPACE
005640        IF   130-ODNR > 1
005650                   GO TO 0350P
005660        ELSE   IF   W01004R = "NIL"   GO TO 0350P.
005680    IF  "XX" = W01020R OR W01021R OR W01022R
005690        IF   0 = 133-PXCPF AND 133-PXCPS AND 133-PXCPT
005690                        GO TO 9390F.
005710    IF  "XX" = W01023R OR W01024R OR W01025R
005720        IF   0 = 133-PXCBF AND 133-PXCBS AND 133-PXCBT
005720                        GO TO 9400F.
005740    IF   W01020-22 NOT = SPACE
005750        IF   "XX" = W01020R OR W01021R OR W01022R
005760                       GO TO 0310P  //Backward Branch
005770        ELSE     GO TO 0320P.  //Backward Branch
005780    IF   0 = 133-PXCPF AND 133-PXCPS AND 133-PXCPT  GO TO 0320P.
005800    IF   130-ODNR > 1
005820        COMPUTE WS-CC11 =
005820        WS-CC11 + 133-PXCPF + 133-PXCPS + 133-PXCPT
005820                        GO TO 0310P.  //Backward Branch
005840    IF   1 = WS-SWP OR WS-SWD  GO TO 0310P.  //Backward Branch
```

The programmer‘s Dream



Procedural Code Pre- & Post Conditions



Code Reengineering Process

Transforms the Code in several optional Steps

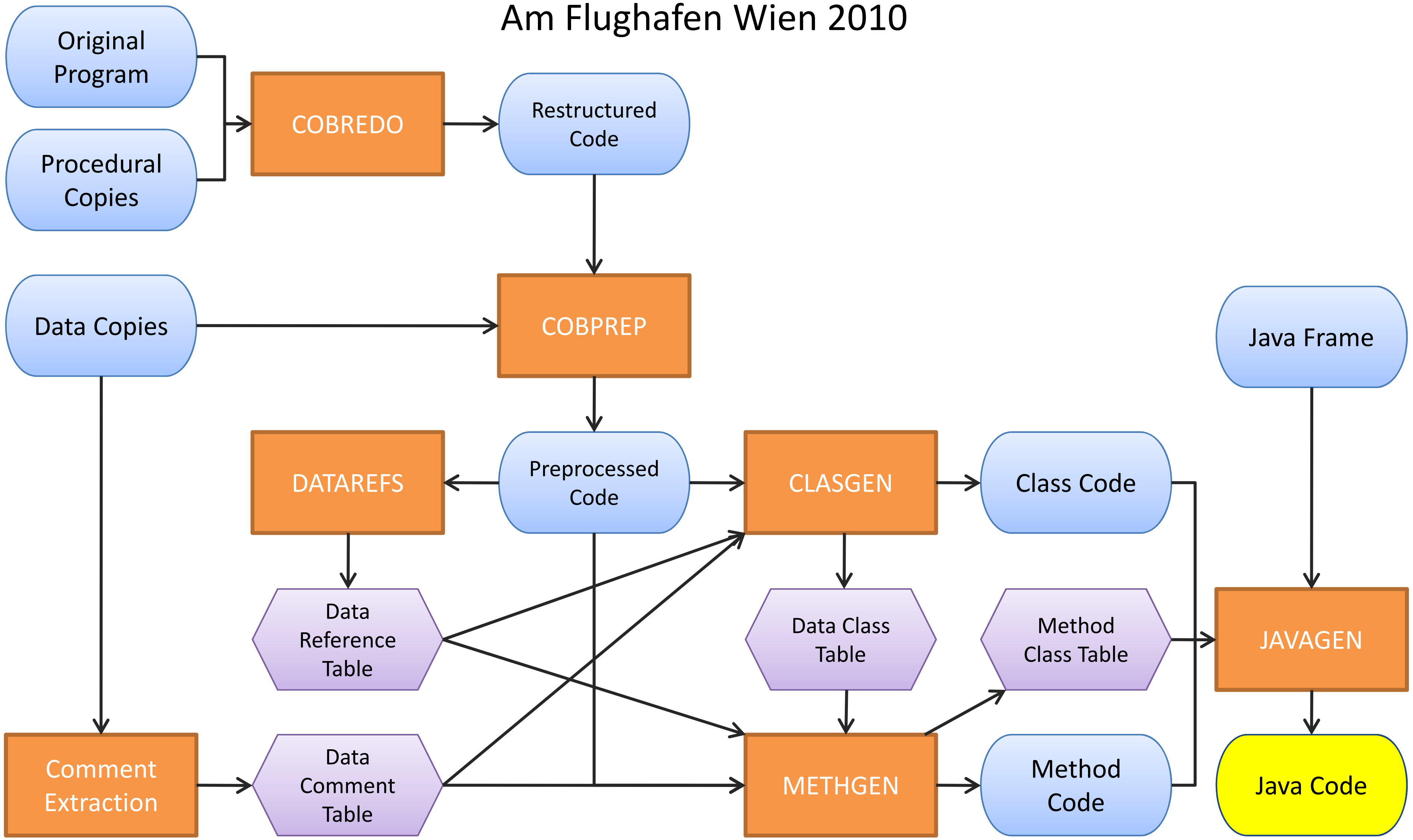
Figure 2: Code Purification

TUD

MIG-18

COBOL to Java Transformation Process

Am Flughafen Wien 2010



Converted Data Structures

```
public  class WRK_RRE_000                 extends COBOLObject {
/**Define static class Object                            <br>*/

    private static  WRK_RRE_000                  instance = null;
//õõ This is a place holder for potential methods
    public  static  char[]    RRE_000;
    private WRK_RRE_000(Object program) {
        RRE_000 = new char[282 ];
        RRE_000 = this.xSpaces(282 ).toCharArray();
        RRE1 = (RRE1)program;
        initWRK_RRE_000();     }
/*<Attributes>                                           <br>*/
/*<Attr name = "RRE-000" type = "Struct" pos = "0000" lng = "0282" comment = "undefined"/> <br> */
/**Get Attribute Method                                  <br>*/
    public String getRRE_000() {
        return  getString(RRE_000,0,282);     }
/**Set Attribute Method                                  <br>*/
    public void setRRE_000(String inStr) {
        setAsChar(RRE_000,inStr,0,282);     }
/*<Attr name = "RRE-FMT" type = "X(04)" pos = "0000" lng = "0004" comment = "undefined"/> <br> */
/**Get Attribute Method                                  <br>*/
    public String getRRE_FMT() {
        return  getString(RRE_000,0,4);     }
/**Set Attribute Method                                  <br>*/
    public void setRRE_FMT(String inStr) {
        setAsChar(RRE_000,inStr,0,4);     }
/*<Attr name = "RRE-FJJ" type = "9(02)" pos = "0055" lng = "0002" comment = "undefined"/> <br> */
/**Get Attribute Method                                  <br>*/
    public int    getRRE_FJJ() {
        return  getInteger(RRE_000,55,2);    }
/**Set Attribute Method                                  <br>*/
    public void setRRE_FJJ(int invalue) {
        setAsChar(RRE_000,abs(invalue),55,2);    }
/*<Attr name = "RRE-FMM" type = "9(02)" pos = "0058" lng = "0002" comment = "undefined"/> <br> */
/**Get Attribute Method                                  <br>*/
    public int    getRRE_FMM() {
        return  getInteger(RRE_000,58,2);    }
/**Set Attribute Method                                  <br>*/
    public void setRRE_FMM(int invalue) {
        setAsChar(RRE_000,abs(invalue),58,2);   }
```

Converted Procedural Code

```
//* CUTE-Geraete haben ihre spez. User-ID
//  IF TXL1 = "U"
   if (IAB6.INPUT.INP_RTX_STORAGE.getTXL1().compareTo("U") == 0) {
//    MOVE TXL1-3 TO USR-IDENT USR-ID R223-LID
            IAB6.WORK.WRK_USR_IDENT.setUSR_IDENT(IAB6.INPUT.INP_RTX_STORAGE.getTXL1_3());
            IAB6.WORK.WRK_IAB6_WORKING.setUSR_ID(IAB6.INPUT.INP_RTX_STORAGE.getTXL1_3());
//     GO XAUT-1
     xNextMethod = " IAB6.WORK.WRK_IAB6_WORKING.IAB6_XAUT_1";
     return xNextMethod;  }
//  END-IF.
//*  suche LID-Anmeldungssatz    (llll----)
//   IF TXL1-2 = "SV" OR TXL1-3 = "C@96@F"
if (IAB6.INPUT.INP_RTX_STORAGE.getTXL1_2().compareTo("SV") == 0 ||
   IAB6.INPUT.INP_RTX_STORAGE.getTXL1_3().compareTo("C@96@F") == 0) {
//       MOVE "FD"       TO R223-LID USR-ID
       this.setR223_LID("FD");
//   ELSE     }
    else {
//       MOVE TXILID     TO R223-LID
       this.setR223_LID(IAB6.INPUT.INP_RTX_STORAGE.getTXILID());
//       MOVE ALL "-"    TO R223-MSID
       this.setR223_MSID("-"[*]);
//       FIND ANY R223AUTY
       DB_STATUS = IDSDB.IDS_R223AUTY.find(R223AUTY, "ANY");
//*    Dieser Satz enthaelt die User-ID
//       IF DB-STATUS NOT ZERO
        if (!(IAB6.WORK.WRK_IAB6_WORKING.getDB_STATUS().matches(Zeros))) {
//            GO 9875F
           xNextMethod = " IAB6.WORK.WRK_IAB6_WORKING.IAB6_9875F";
           return xNextMethod;
//       ELSE          }
       else {
//           GET
          DB_STATUS = IDSDB.IDS_R223AUTY.getnext(R223AUTY, this);
//           MOVE R223-SA1  TO USR-ID
          IAB6.WORK.WRK_IAB6_WORKING.setUSR_ID(this.getR223_SA1());
//       END-IF          }
```

Structural Documentation in XML

```
- <Class name="IDS_R208LIDT" type="IDSDB">
   <Object name="R208LIDT" type="char[124 ]" />
-  <Attributes>
     <Attr name="R208LIDT" type="Struct" pos="0000" lng="0124" comment="undefined" />
     <Attr name="R208-LIDT" type="X(4)" pos="0000" lng="0004" comment="undefined" />
     <Attr name="R208-SA1" type="X(120)" pos="0004" lng="0120" comment="undefined" />
  </Attributes>
-  <AccessMethods>
     <Method name="storeR208LIDT" type="access" />
     <Method name="findR208LIDT" type="access" />
     <Method name="getnextR208LIDT" type="access" />
     <Method name="modifyR208-LIDT" type="access" />
     <Method name="eraseR208LIDT" type="access" />
  </AccessMethods>
-  <ProcessingMethods>
-    <Method name="DAU3_PRT_RP" type="processing">
-      <Parameters>
                <Param use="Input" name="R208_LIDT" class="IDS_R208LIDT" type="String"/>
                <Param use="Input" name="TXL1_3" class="INP_RTX_STORAGE" type="String"/>
                 <Param use="Output" name="R208_LIDT" class="IDS_R208LIDT" type="String"/>
                 <Param use="InOut" name="R208LIDT" class="IDS_R208LIDT" type="String"/>
                 <Param use="Input" name="DB_STATUS" class="WRK_DAU3_WORKING" type="int"/>
                 <Param use="Input" name="R208_SA1" class="IDS_R208LIDT" type="String"/>
                 <Param use="Output" name="DB_208_SA1" class="WRK_RR_208_SA1" type="String"/>
                 <Param use="Output" name="DESTINATION_ID_3" class="OUT_MSG"/>
                 <Param use="Input" name="DESTINATION_ID_1" class="OUT_MSG" type="char"/>
       </Parameters>
-  <Successors>
         <NextMethod name="DAU3:IDSDB.IDS_R208LIDT.DAU3_PRT_RP_Z" type="GOTO" />
         <NextMethod name="DAU3:IDSDB.IDS_R208LIDT.DAU3_PRT_RP" type="GOTO" />
   </Successors>
  </Method>
 </Class>
```

Ergebnisse des Wiener Flughafen Migration Projektes

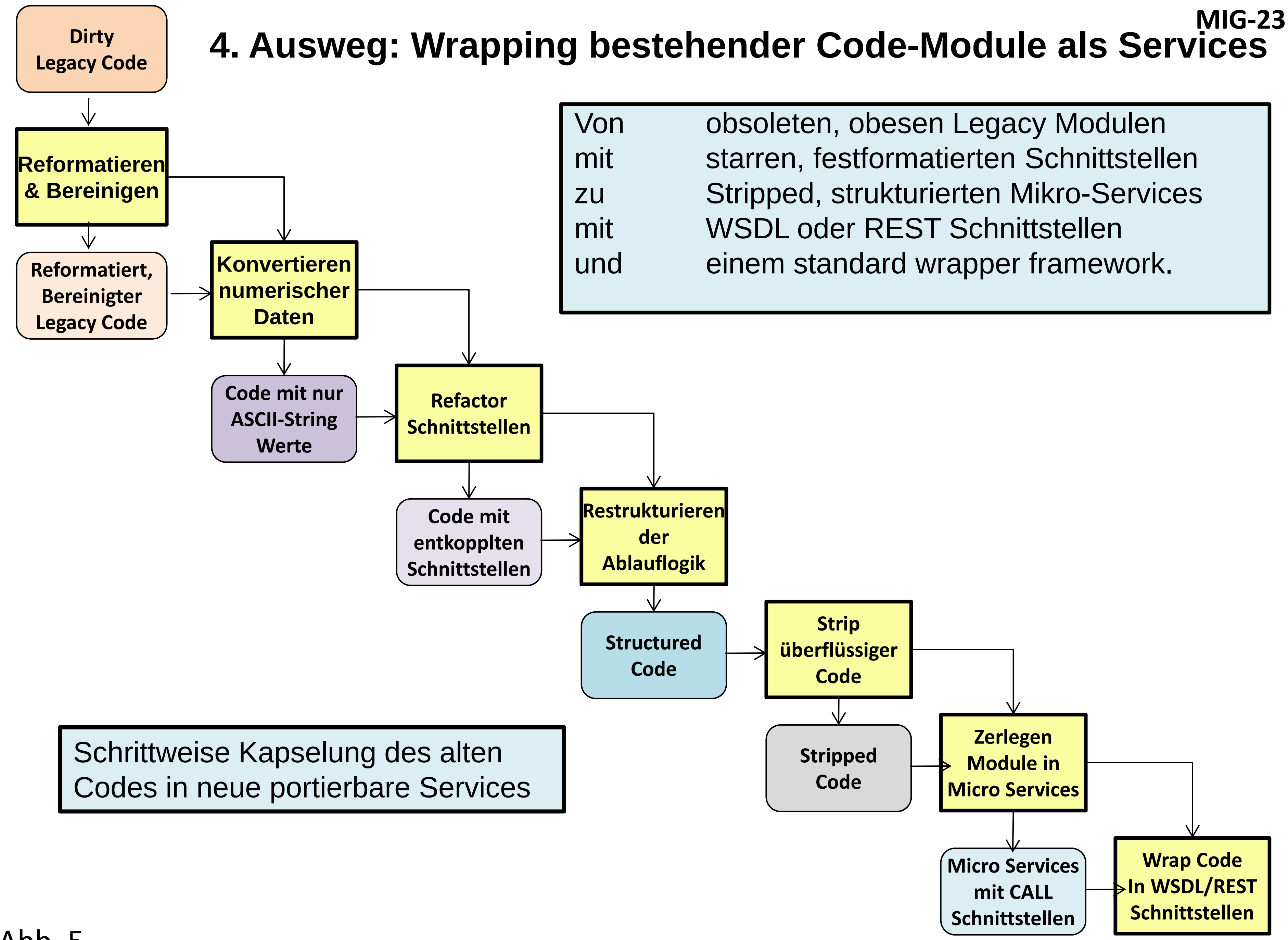
Die alte Source besteht aus IDS Datenbankschemen, Subschemen bzw. Sichten, Untermodule und Hauptprogramme. Sie haben zusammen:
• 1721 Sourcen
• 994.541 echte Codezeilen
• 81.767 Data-Points
• 20.486 Function-Points

Die neue Source besteht aus Java Klassen, JSP Skripten und SQL Datenbankschemen. Sie haben zusammen:
• 3756 Sourcen
• 312.571 echte Codezeilen
• 95.286 Data-Points
• 18.942 Function-Points

Die neuen Systeme haben also doppelt so viele Sourcen mit nur einem Drittel der Codezeilen, die das alte System hat.
Durch die Restrukturierung der Altprogramme hat sich die Zeilenzahl allerdings verdoppelt.
Dieser Schritt war aber unerlässlich um den Code zu entwirren.
Die Zahl der Data-Points und Function-Points sind in etwa gleich.
Die neue Software hat 16% mehr Data-Points. Die alte Software hat dagegen 8% mehr Function-Points.

4. Ausweg: Wrapping bestehender Code-Module als Services

Von	obsoleten, obesen Legacy Modulen
mit	starren, festformatierten Schnittstellen
zu	Stripped, strukturierten Mikro-Services
mit	WSDL oder REST Schnittstellen
und	einem standard wrapper framework.


Dirty Legacy Code
Reformatieren & Bereinigen
Reformatiert, Bereinigter Legacy Code
Konvertieren numerischer Daten
Code mit nur ASCII-String Werte
Refactor Schnittstellen
Code mit entkopplten Schnittstellen
Restrukturieren der Ablauflogik
Structured Code
Strip überflüssiger Code
Stripped Code
Zerlegen Module in Micro Services
Micro Services mit CALL Schnittstellen
Wrap Code In WSDL/REST Schnittstellen

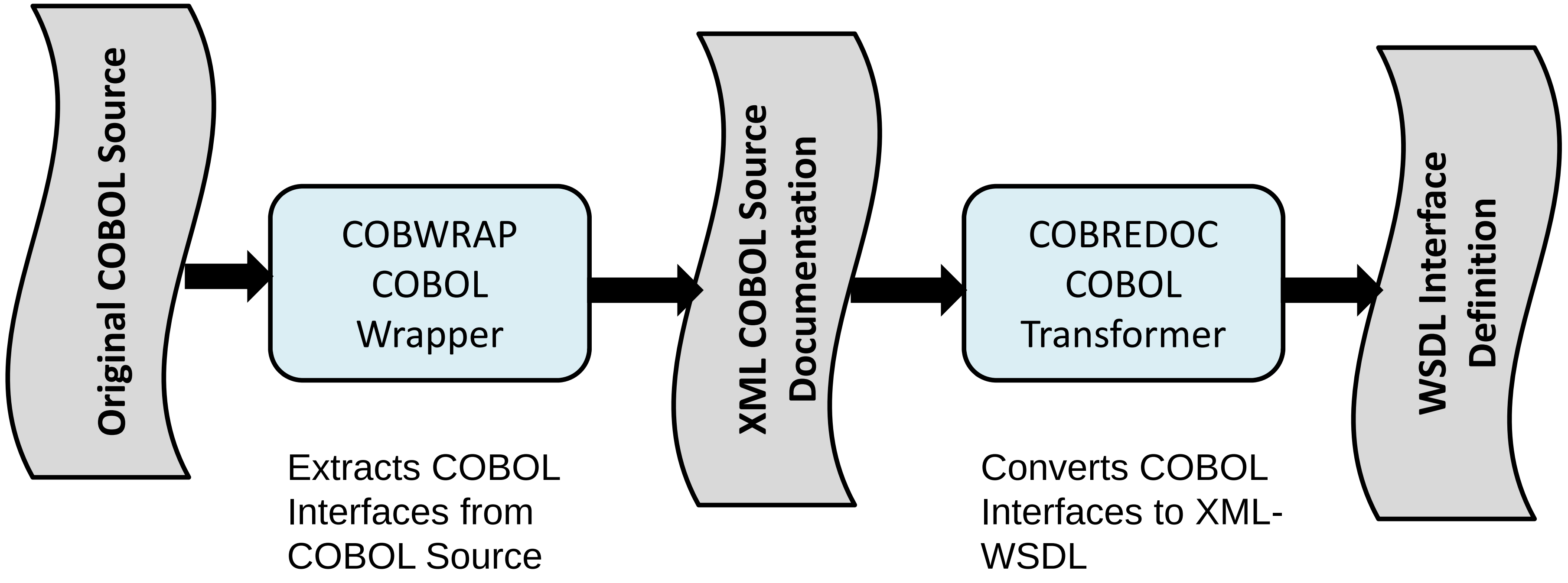
Schrittweise Kapselung des alten Codes in neue portierbare Services

Abb. 5

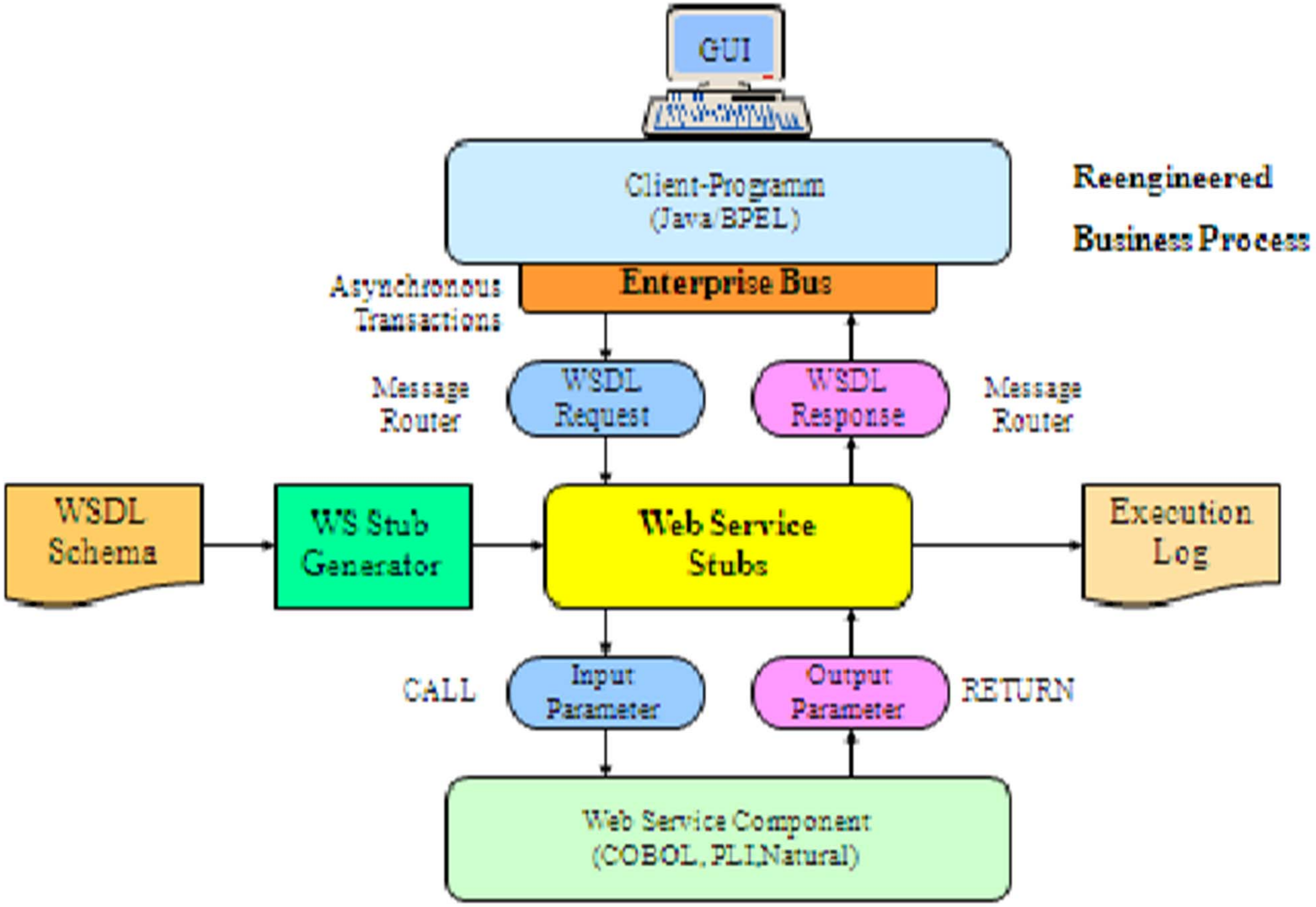
Steps to Wrapping a Micro Service

- Isolate the Code to be used as a Service from the Code around it.
- Replace the method interfaces with service interfaces.
- Adjust the interfaces to the requirements of the Service Users.
- Test the service operations with generated service requests
- Evaluate the suitability of the Service.

From native COBOL to a WSDL Interface Definition


Original COBOL Source
COBWRAP
COBOL
Wrapper
XML COBOL Source Documentation
COBREDOC
COBOL
Transformer
WSDL Interface Definition
Extracts COBOL Interfaces from COBOL Source
Converts COBOL Interfaces to XML-WSDL

Integrating wrapped legacy components



5. Ausweg: Re-implementierung des prozeduralen Codes

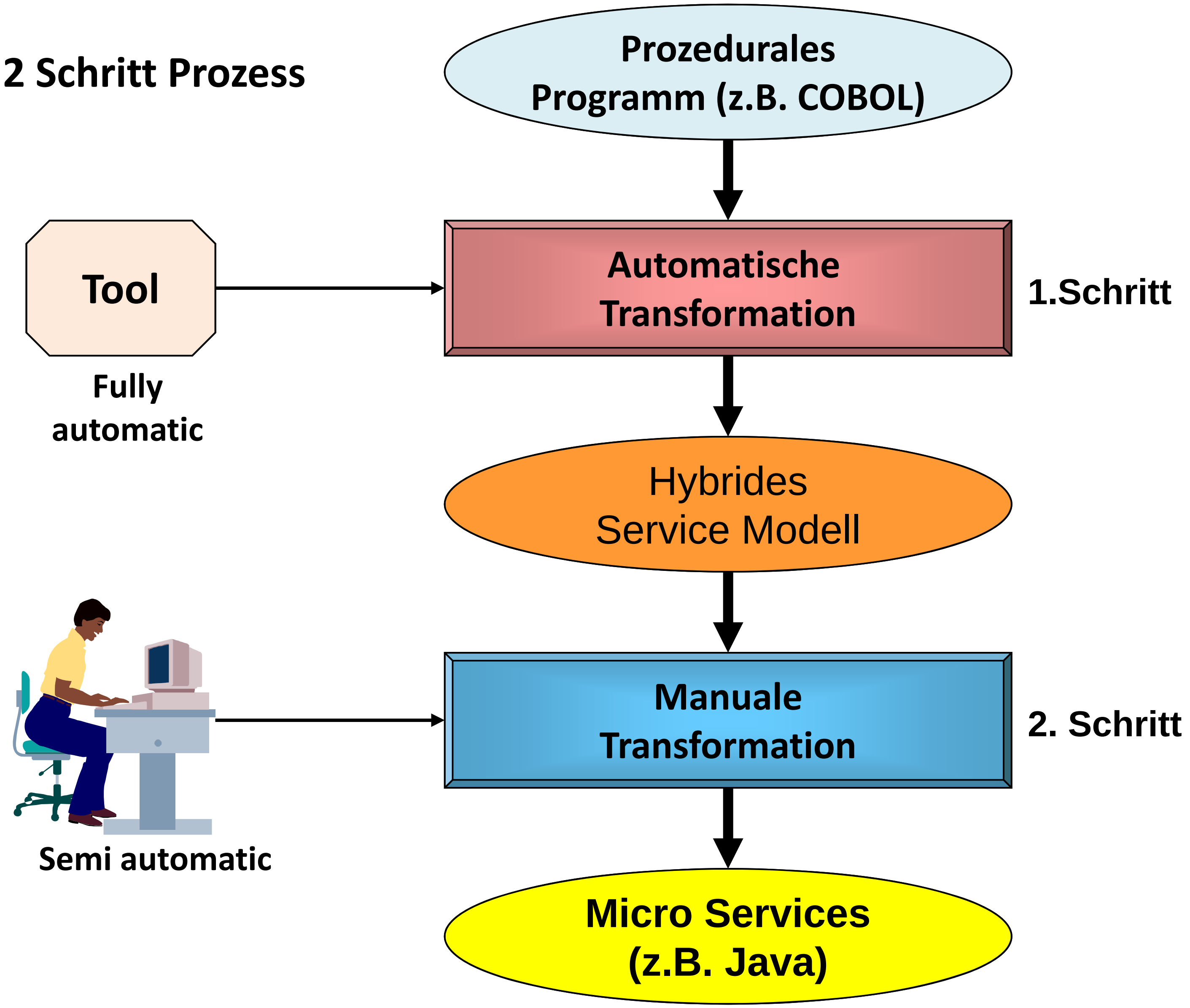
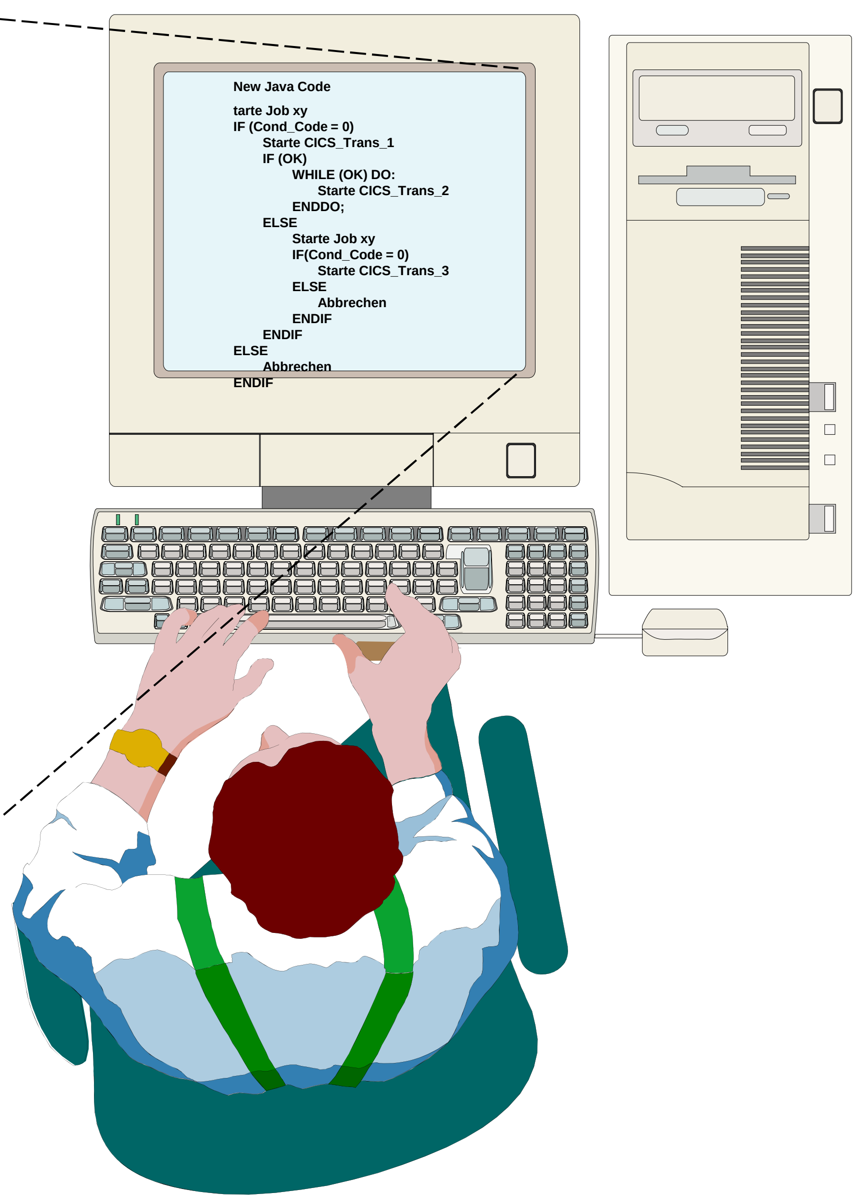


Abb. 7

Rewriting Old Code in a new Language

Old Pseudo Code

```
Start Job xy
IF (Cond_Code = 0)
        Start Trans_1
        IF (OK)
                WHILE (OK) DO:
                        Start Trans_2
                ENDDO;
        ELSE
                Start Job_4
                IF(Cond_Code = 0)
                        Start Trans_3
                ELSE
                        terminate
                ENDIF
        ENDIF
ELSE
        terminate
ENDIF
```



New Java Code

```
tarte Job xy
IF (Cond_Code = 0)
        Starte CICS_Trans_1
        IF (OK)
                WHILE (OK) DO:
                        Starte CICS_Trans_2
                ENDDO;
        ELSE
                Starte Job xy
                IF(Cond_Code = 0)
                        Starte CICS_Trans_3
                ELSE
                        Abbrechen
                ENDIF
        ENDIF
ELSE
        Abbrechen
ENDIF
```

Original legacy COBOL Code

```
*     READ ORDERS UNTIL END OF ORDER-FILE
       READ-ORDERS.                    // Here the program reads in the next customer order
           READ ORDER-FILE
               AT END
                 GO TO TERMINATION
           END-READ.
      **************************************************************
      * READ CUSTOMER-DATA WITH KEY = CUST-NO
       READ-CUSTOMER.                  // Here the program selects the customer who placed the order.
           MOVE ZERO TO ERROR-TYPE.
           MOVE CUST-NO IN ORDER-RECORD TO CUST-KEY.
           READ CUSTOMER-FILE
               INVALID KEY MOVE 1 TO ERROR-TYPE
                          GO TO REPORT-ERROR.
           MOVE 0 TO POS.
**************************************************************
 * PROCESS ORDER ITEMS FROM 1 TO 9  OR UNTIL ITEM-NO = 999
       PROCESS-ORDER.        // Here the program processes one order item after the other.
           ADD 1 TO POS.
           IF POS > 9  OR ITEM-NO IN ORDER-RECORD (POS) = 9
                    SUBTRACT 1 FROM POS
                    GO TO PRINT-SUMMARY.
      *  READ ARTICLE DATA WITH KEY = ART-NO
         READ ARTICLE-FILE. // Here the program selects the ordered article.
               INVALID KEY MOVE 2 TO ERROR-TYPE
                          GO TO REPORT-ERROR.
      * CHECK IF QUANTITY IS SUFFICIENT AND DEDUCT ORDER FROM STOCK
           IF ITEM-QUAN IN ORDER-RECORD (POS) >    // Here the program checks the article amount
             ART-QUAN IN ARTICLE-RECORD
                       MOVE 3 TO ERROR-TYPE
                       GO TO WRITE-OPEN-POSITIONS
           ELSE   // Here the program deducts the amount ordered from the article amount and dispatches
             SUBTRACT ITEM-QUAN IN ORDER-RECORD (POS) FROM
                     ART-QUAN IN ARTICLE-RECORD
                     REWRITE ARTICLE-RECORD
                     GO TO WRITE-DISPATCH
```

Order Entry translated to Java

```
// The same Code rewritten in Java
public class CustomerOrder {    // Here the class variables are declared.
                private int OrderNumber;
                private int OrderAmount;
                private Date OrderDate;
                private Customer Customer;
                private List<OrderItem> OrderItems;
                private String PaymentInitials;
                private OrderItem OrderItem;
                private CustomerOrder vCustomerOrder;
                private List<OrderItem> oItems = new ArrayList<OrderItem>();
                private Article article;
                private Invoice inv;
                private static SessionFactory sessionFactory;

                public CustomerOrder(CustomerOrder cusOrder){ // Here the class variables are initialized.
                                this.OrderNumber = cusOrder.OrderNumber;
                                this.OrderAmount = cusOrder.OrderAmount;
                                this.OrderDate = cusOrder.OrderDate;
                                this.Customer = cusOrder.Customer;
                                this.OrderItems = cusOrder.OrderItems;
                                this.PaymentInitials = cusOrder.PaymentInitials;
                                this.OrderItem = cusOrder.OrderItem;
                                setCustomerOrder(cusOrder);
                }
protected CustomerOrder getCustomerOrder(int OrderNumber){
                                return this;
                }
```

TUD MIG-31

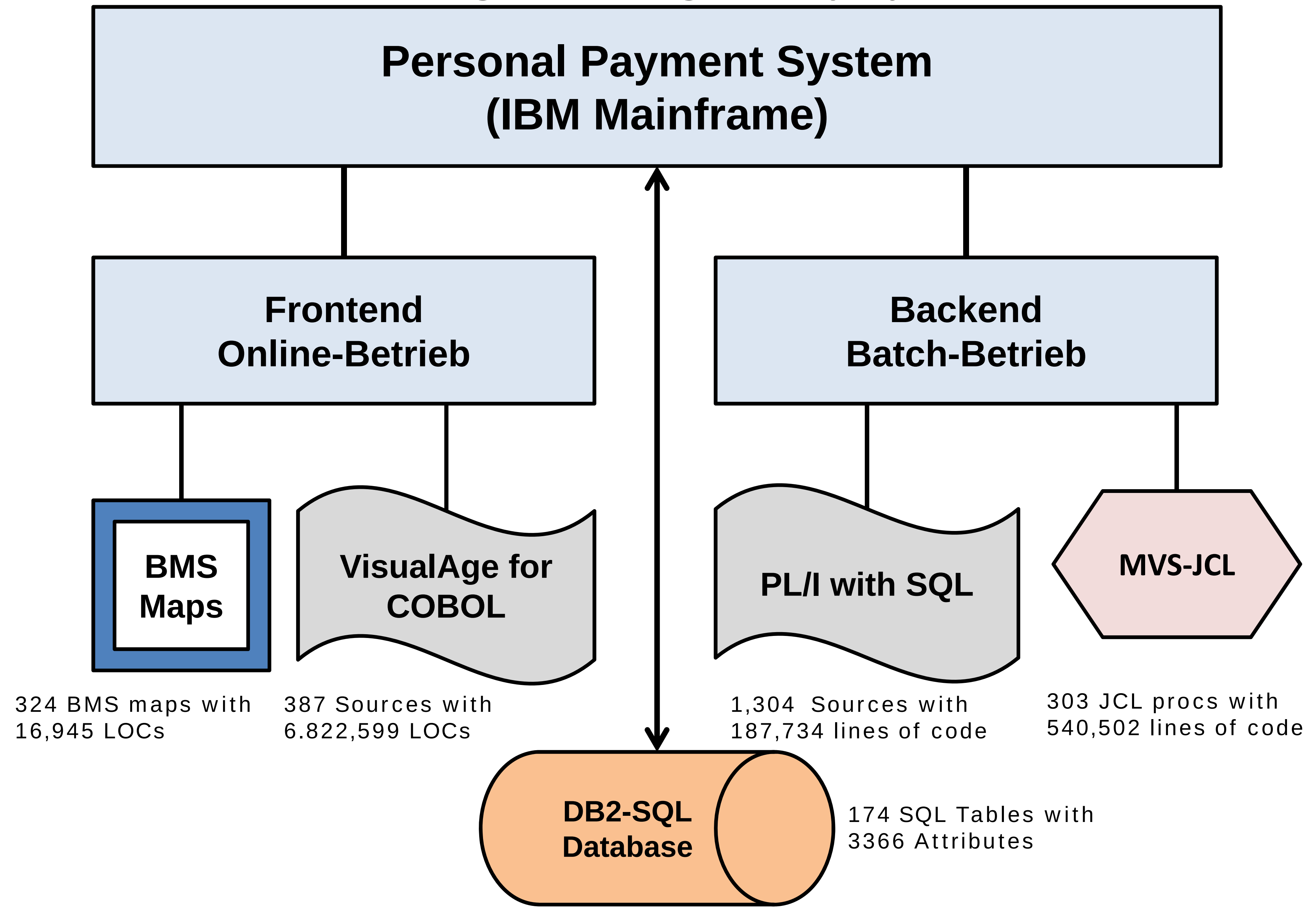
```
                         protected void setCustomerOrder(CustomerOrder CustomerOrder){
                         if (this.Customer.getCustomerStatus(this.Customer).equals("good")) {
                                 // Create Order
                                 vCustomerOrder = CustomerOrder;
                                 vCustomerOrder.OrderNumber = CreateNewOrderNumber();
                                 // Iterate over OrderItems
                                 oItems = vCustomerOrder.OrderItems;
                                 for(int i = 0; i < oItems.size(); i++) {
                                   article = oItems.get(i).getArticle();
                                   if (article.getArticleAmount(article) > oItems.get(i).getItemAmount()) {
                                         // Create and Write Object into XML
                                         DispatchItem.createDispatchItem(oItems.get(i));
                                         // Deduct the Amount from an Article
                                         article.deductArticleAmount(article.getArticleNumber()); }
                                         // Check if SupplyOrderItem should be generated
                                         if (article.getArticleAmount(article) <= article.getMinimumAmount()) {
                                                 // Create and Write Object into XML
                                                 SupplyOrderItem.createSupplyItem(oItems.get(i)) ;
                                         } else {
                                                 // Create and Write Object into XML
                                                 BackOrderItem.createBackOrderItem(oItems.get(i));
                                                 // Article is not part of the order because it is not available
                                                 this.deleteArticleFromList(oItems.get(i));
                                         }                                                   }
                                 vCustomerOrder.OrderItems = oItems;
                                 // Save CustomerOrder in DB with e.g. Hibernate
                                 saveOrder(vCustomerOrder);
                                 System.out.println("Order fullfilled");
                         } else {
                                 rejectOrder(CustomerOrder.OrderNumber);     }
                 }
```

Sample 3: Manually re-implemented COBOL code

TUD

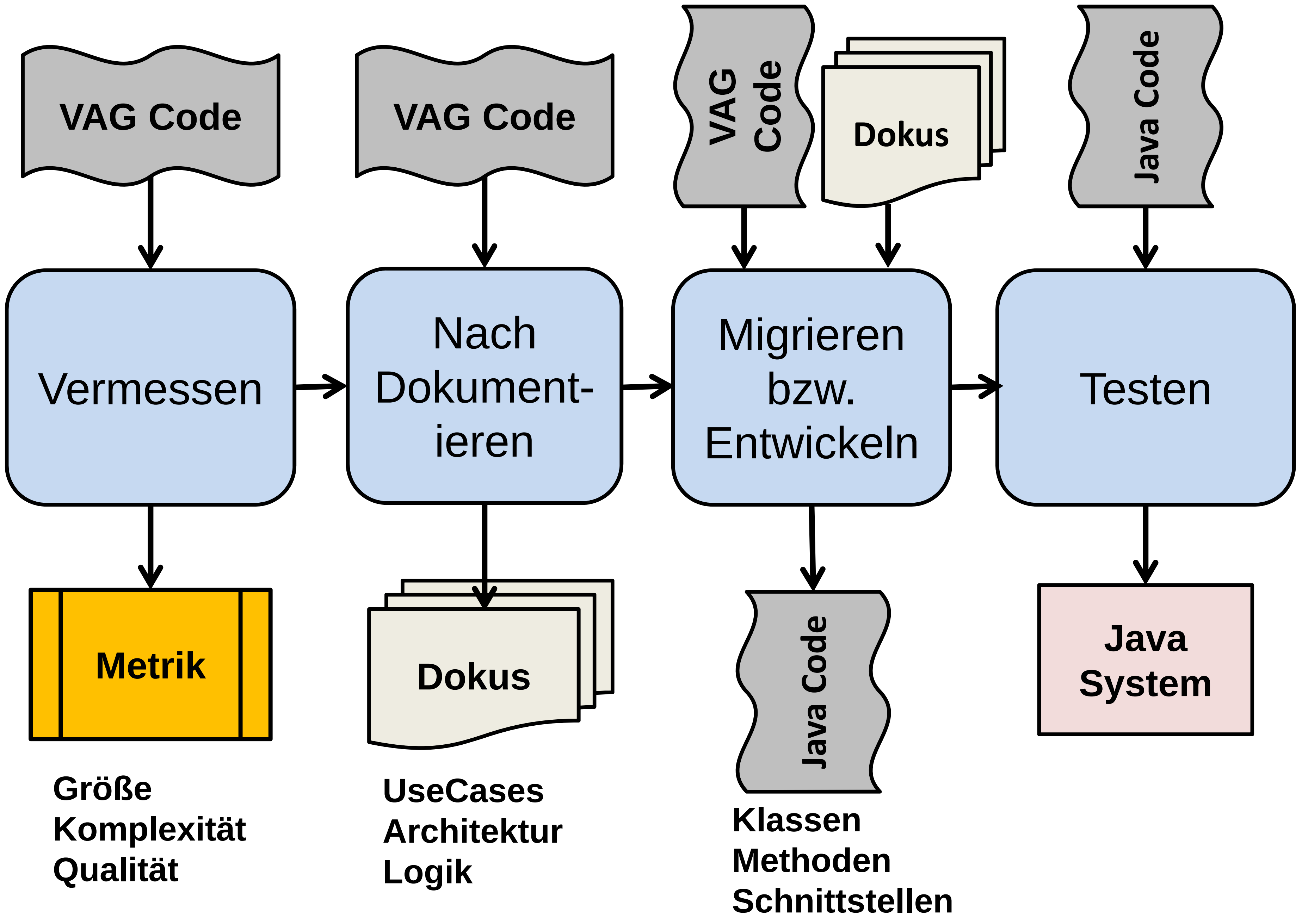
Das Burgenland Migrationsprojekt

MIG-32

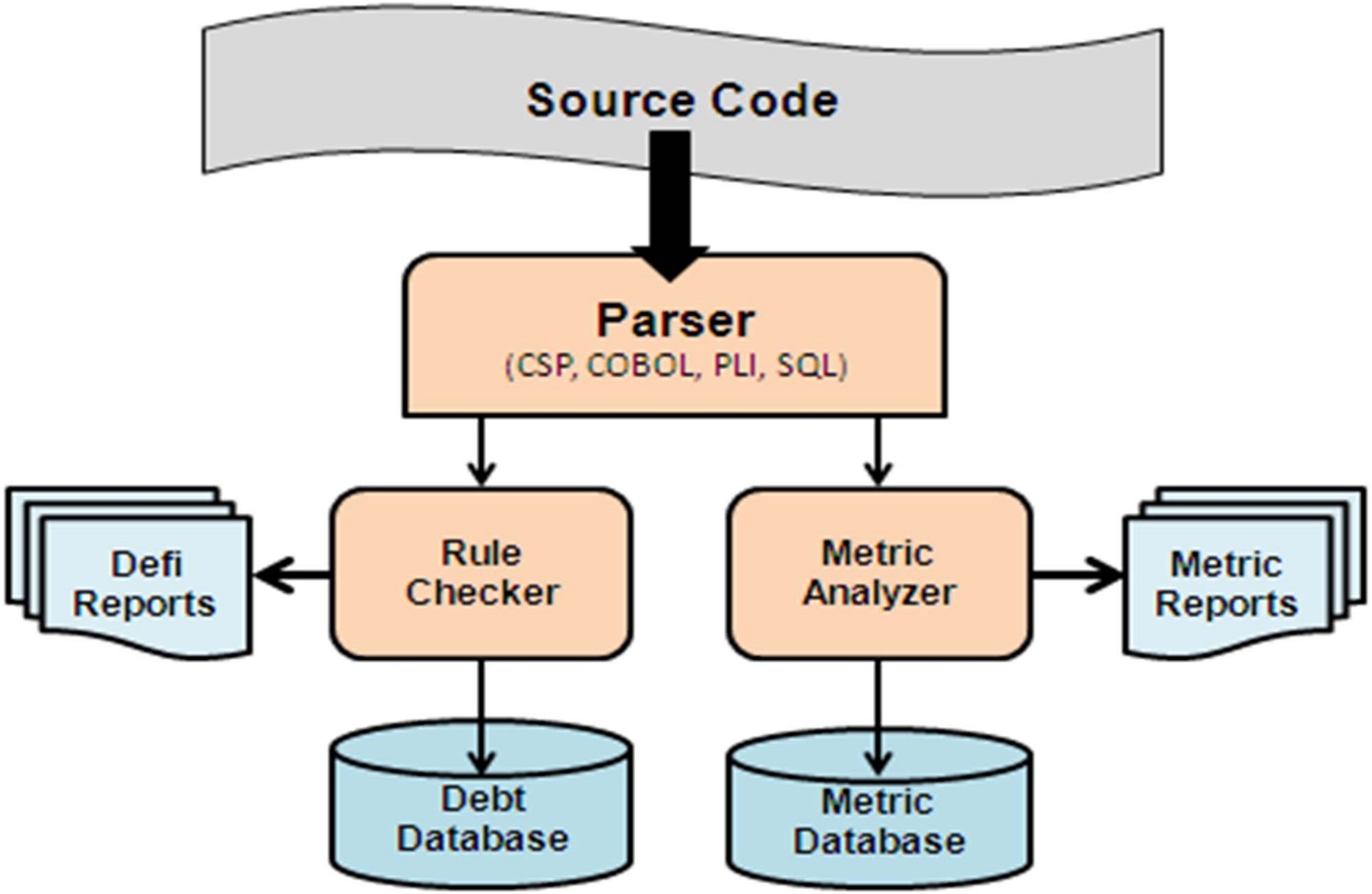


Personal Payment System (IBM Mainframe)
Frontend Online-Betrieb
Backend Batch-Betrieb
BMS Maps
VisualAge for COBOL
PL/I with SQL
MVS-JCL
324 BMS maps with 16,945 LOCs
387 Sources with 6.822,599 LOCs
1,304 Sources with 187,734 lines of code
303 JCL procs with 540,502 lines of code
DB2-SQL Database
174 SQL Tables with 3366 Attributes

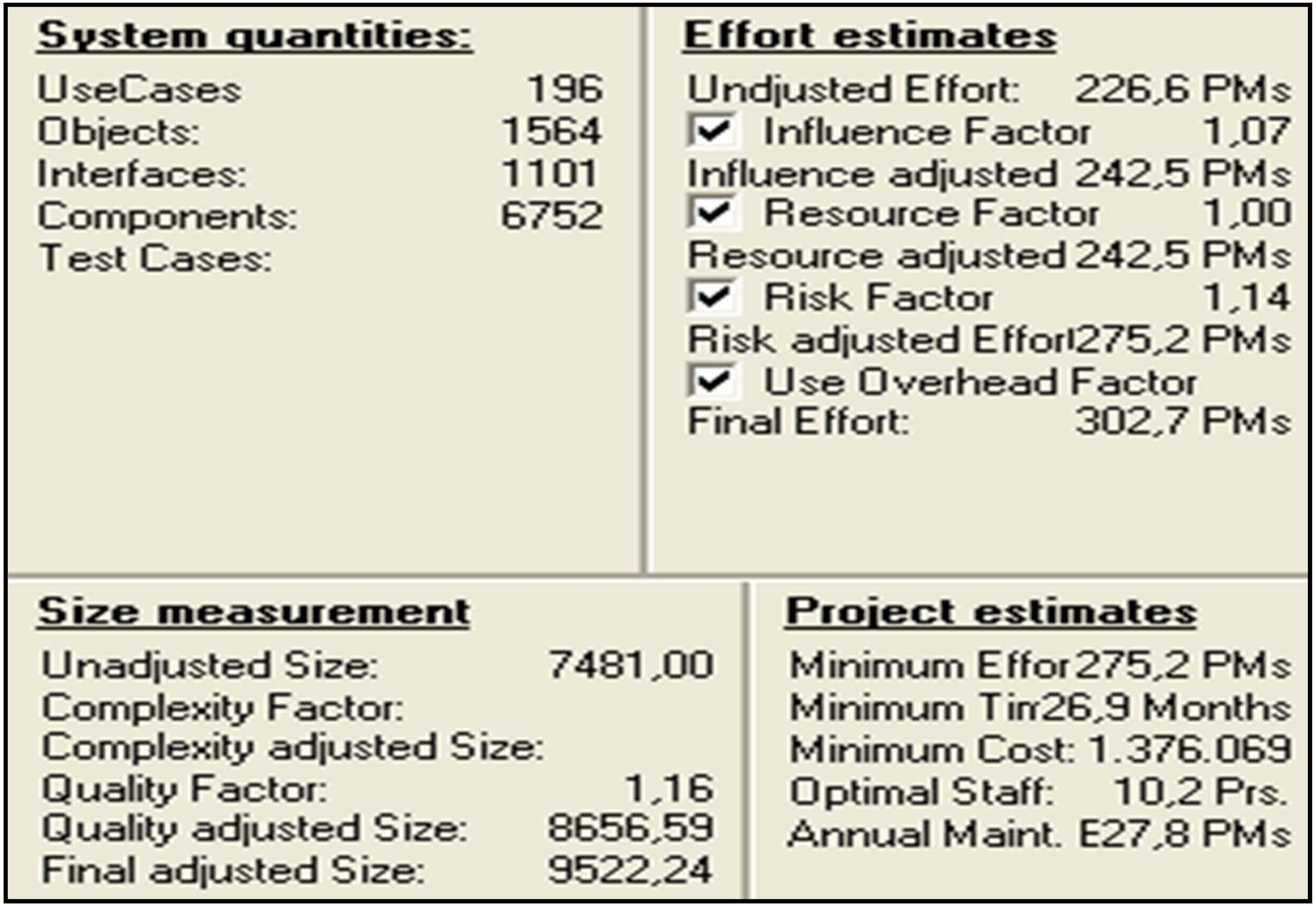
Geplante Migrationsschritte



Automatisierte Analyse des Source-Codes



Frontend Migrationsschätzung nach Function-Point



System quantities:

UseCases	196
Objects:	1564
Interfaces:	1101
Components:	6752
Test Cases:	

Effort estimates

Undjusted Effort:		226,6 PMs
✓	Influence Factor	1,07
Influence adjusted		242,5 PMs
✓	Resource Factor	1,00
Resource adjusted		242,5 PMs
✓	Risk Factor	1,14
Risk adjusted Effor		275,2 PMs
✓	Use Overhead Factor	
Final Effort:		302,7 PMs

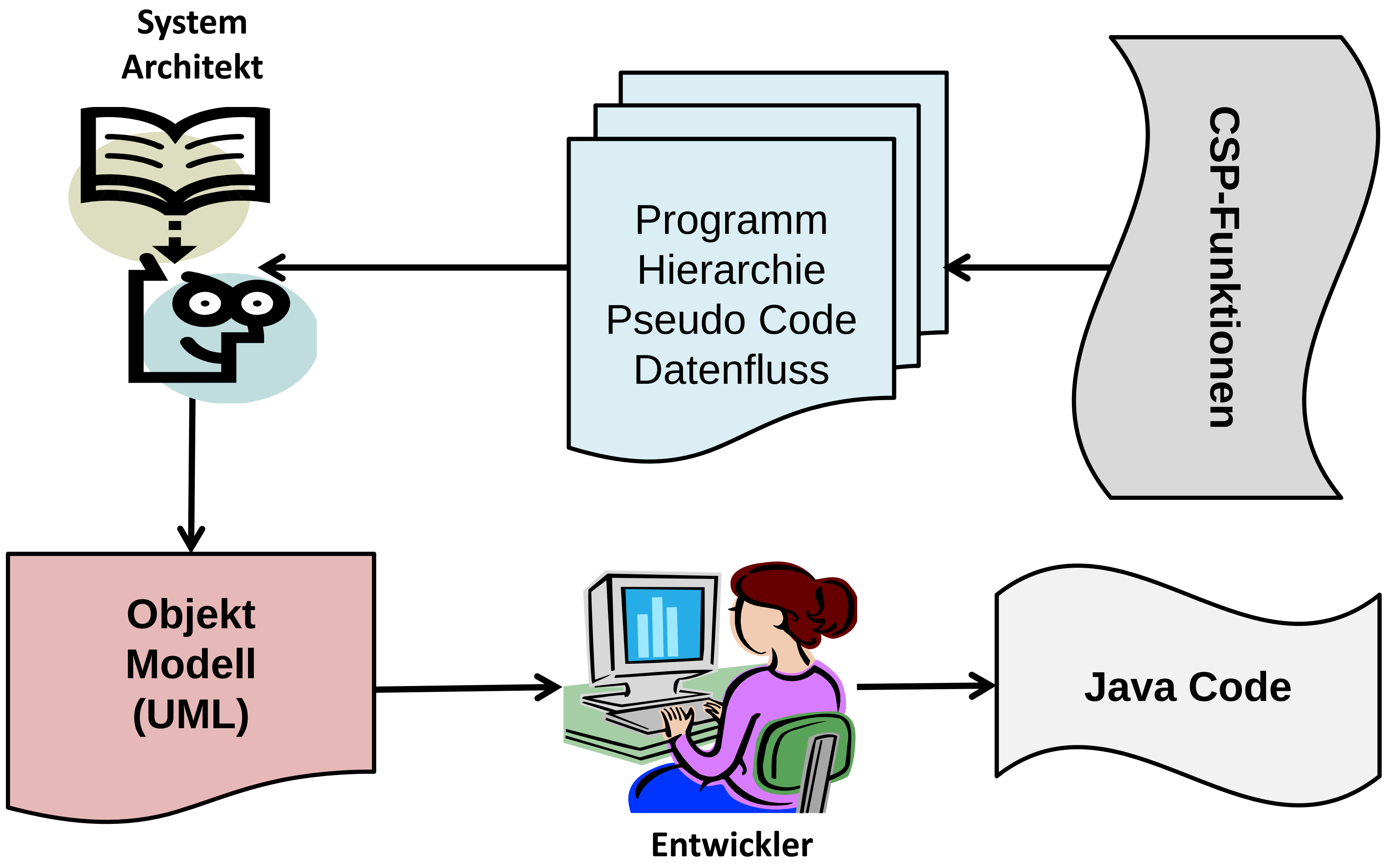
Size measurement

Unadjusted Size:	7481,00
Complexity Factor:	
Complexity adjusted Size:	
Quality Factor:	1,16
Quality adjusted Size:	8656,59
Final adjusted Size:	9522,24

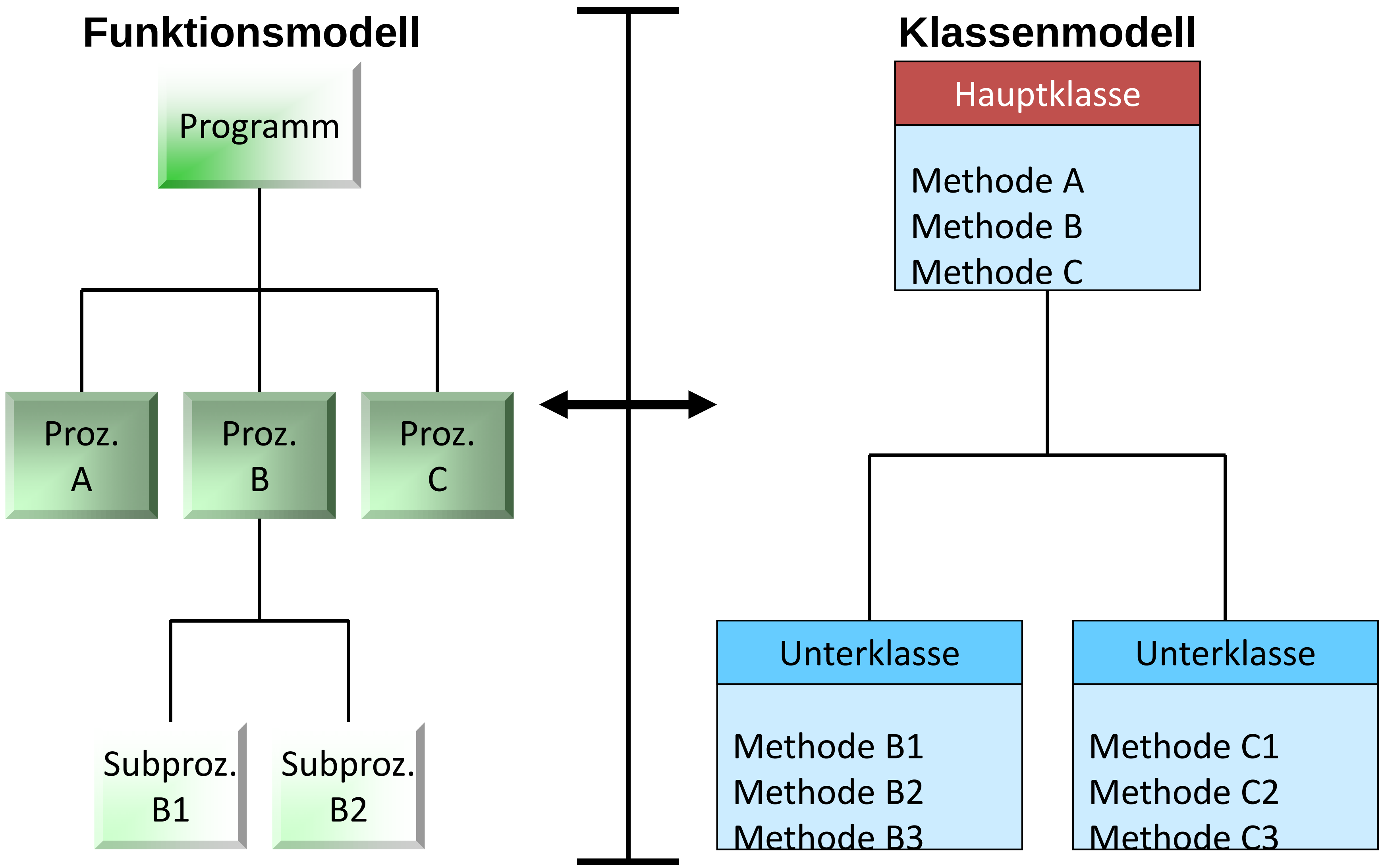
Project estimates

Minimum Effor	275,2 PMs
Minimum Tin	26,9 Months
Minimum Cost:	1.376.069
Optimal Staff:	10,2 Prs.
Annual Maint. E	27,8 PMs

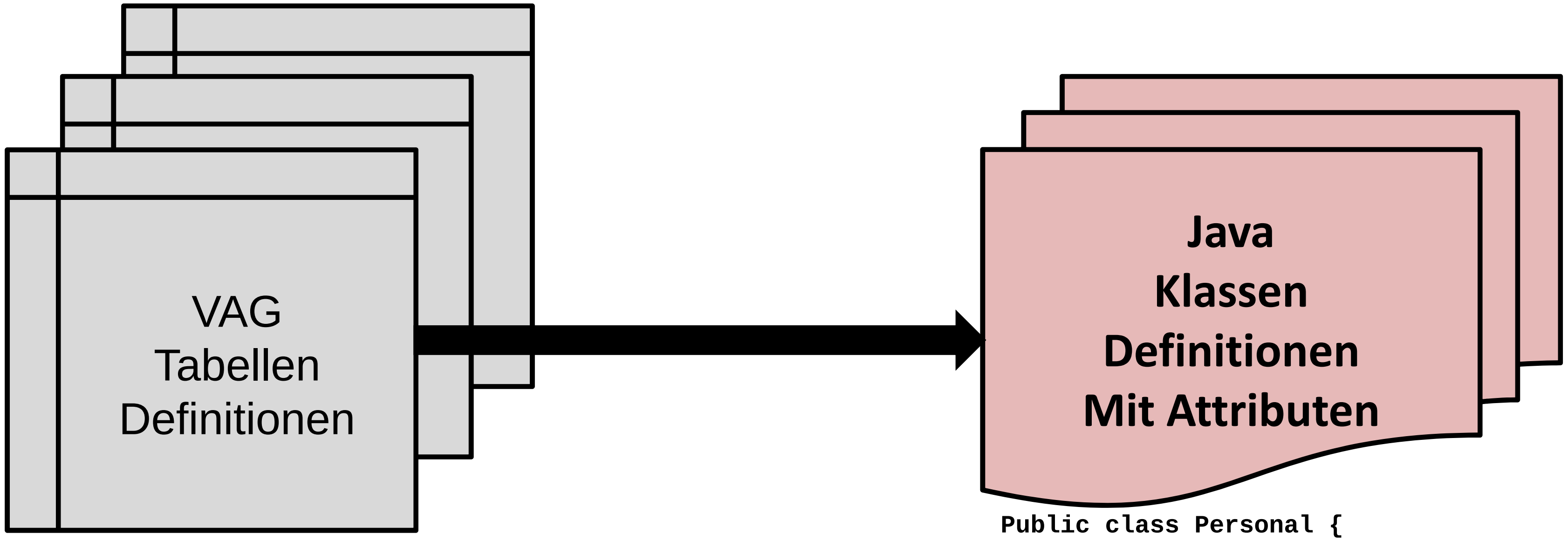
Von VAG Funktionen zu Java Klassen



Überführung der Programmhierarchie in eine Klassenhierarchie



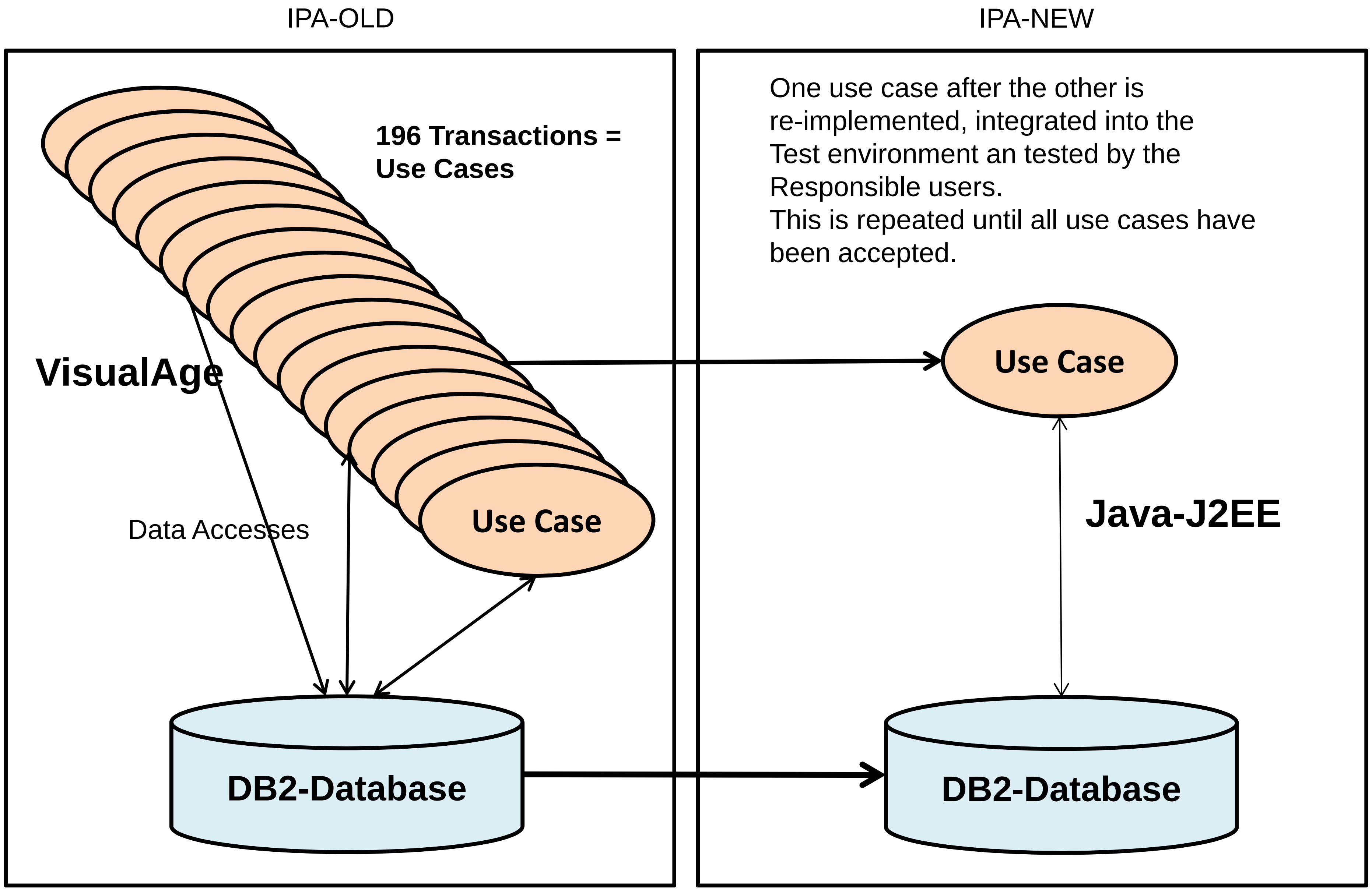
Von VAG Tabellen zu Java Klassen



```
:row.'VONDAT  ' 'D ' ' '  10  00
:row.'BISDAT  ' 'D ' ' '  10  00
:row.'FNAM    ' 'A ' ' '  30  02
:row.'VNAM    ' 'A ' ' '  20  02
:row.'AKGR    ' 'AN' ' '  02  00
:row.'BTIT    ' 'AN' ' '  03  00
:row.'GESCHL  ' 'AN' ' '  01  00
:row.'GEBDAT  ' 'D ' ' '  10  00
:row.'SVNR    ' 'N ' ' '  10  00
:row.'STAAT   ' 'AN' ' '  03  00
:row.'VERWV   ' 'AN' ' '  02  00
:row.'AUSGRU  ' 'AN' ' '  02  00
:row.'KASSE   ' 'AN' ' '  05  00
:row.'BLZ     ' 'N ' ' '  05  00
:row.'KTONR   ' 'N ' ' '  11  00
:row.'NATION  ' 'AN' ' '  03  00
:row.'PLZ     ' 'AN' ' '  05  00
:row.'ORT     ' 'AN' ' '  30  00
:row.'STR     ' 'AN' ' '  30  00
```

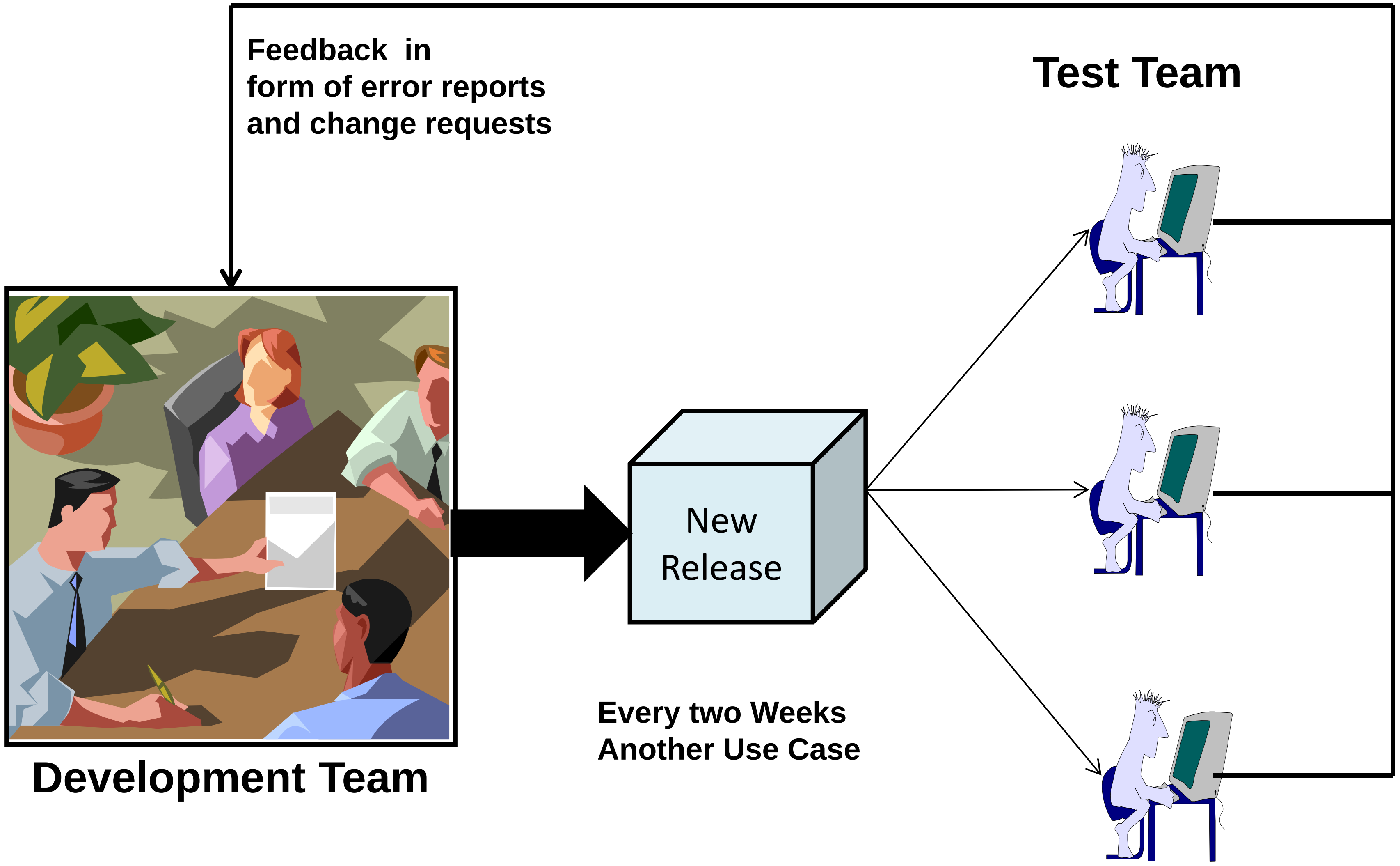
```
Public class Personal {
   Double VonDateum;
   double BisDatum
   string VorName;
   string NachName;
   string AkademischerGrad;
   string Titel;
   char   Geschlect;
   double GeburtsDatum:
   int    Sozialversiherungsnummer;
   string Staatsangehörigkeit;
   string VerwaltungsVersicherung;
   string Auslandsgruppe;
   string Kasse;
   double Bankleitzahl;
   double KontoNummer;
   string Nationalitaet;
   string Postleitzahl;
   string Ort;
   string Strasse;
} // Personenbeschreibung
```

Iterative Delivery- Each Use Case is a Release

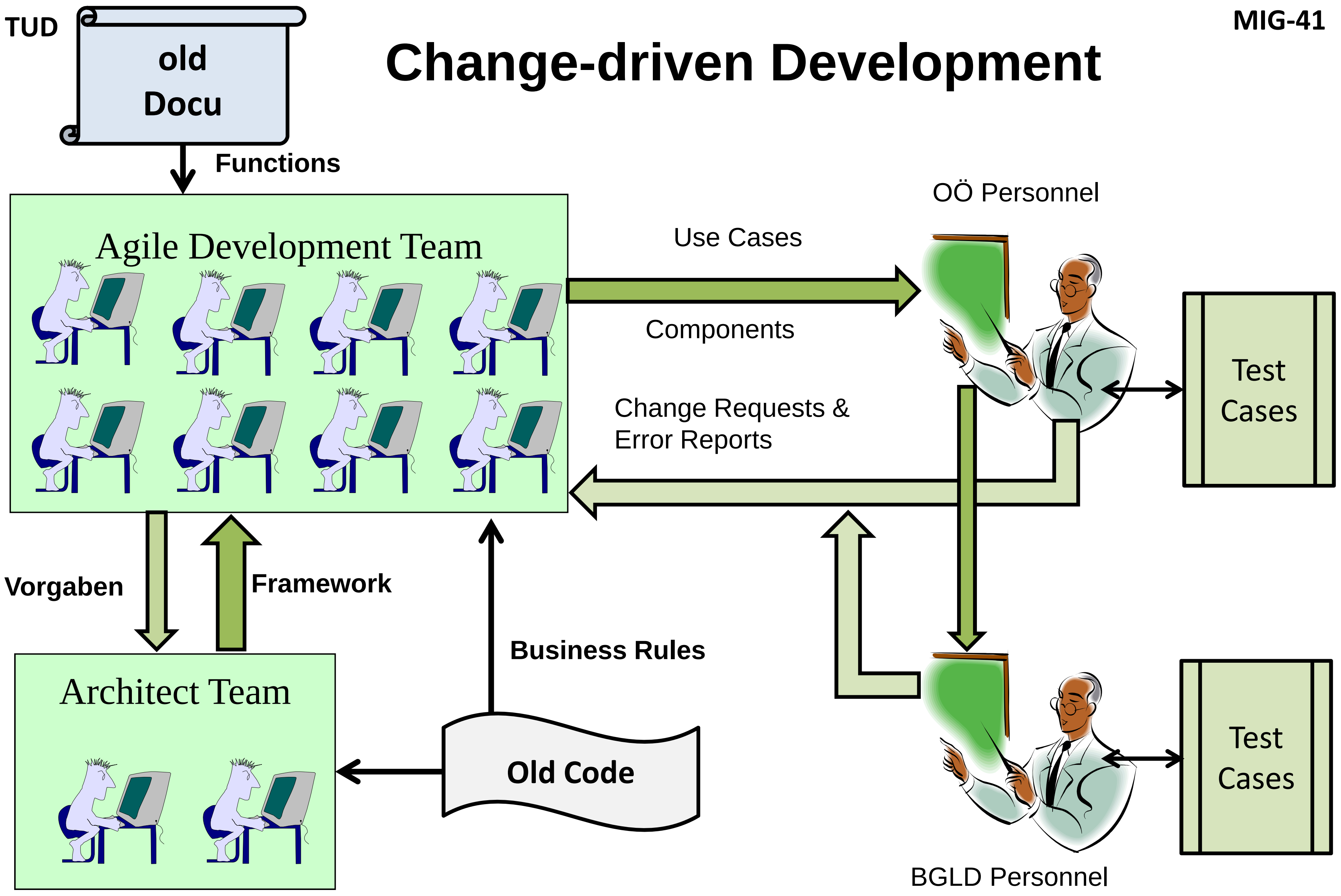


One use case after the other is re-implemented, integrated into the Test environment an tested by the Responsible users.
This is repeated until all use cases have been accepted.

Continuous Delivery



Change-driven Development

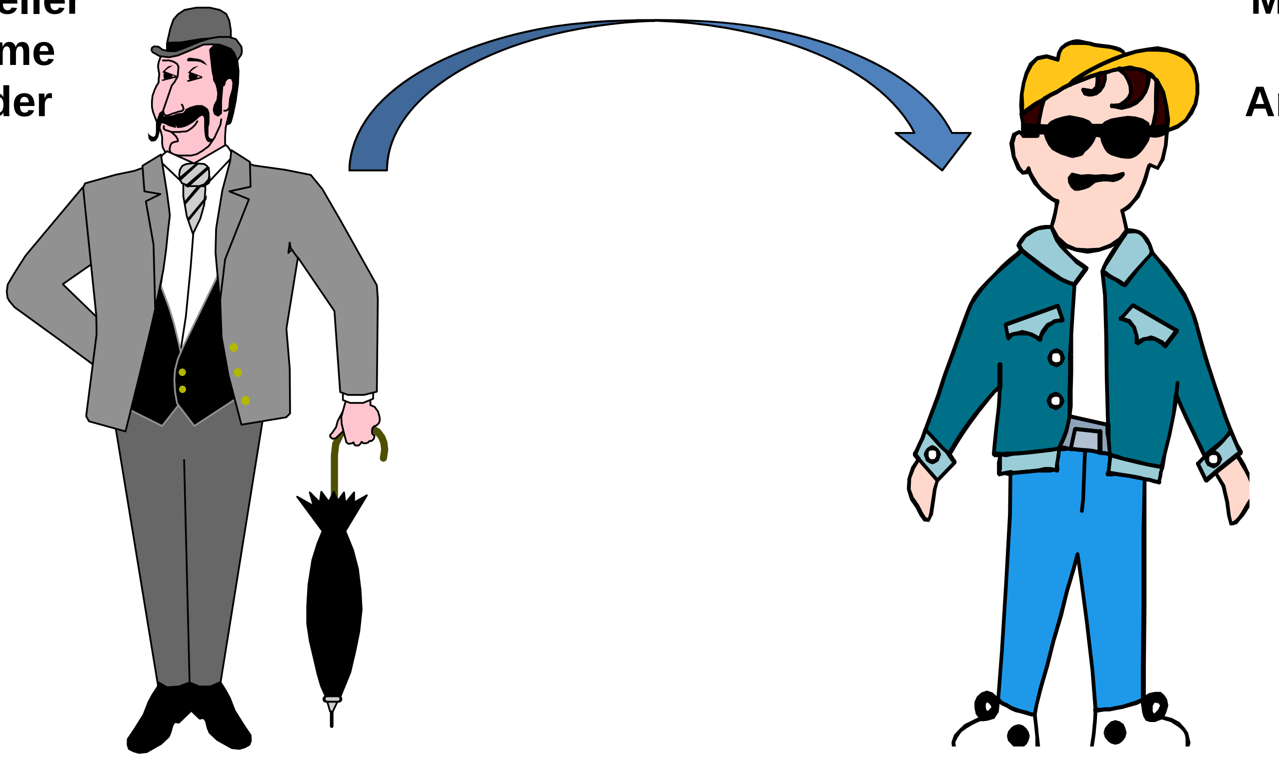


Development by Trial & Error

Migration der Anwender

Traditioneller Mainframe Anwender

Moderner Linux Anwender



Arbeitet mit festformatierten Schirmen.
Bedient die Tastatur wie ein Hackbrett.
Kennt alle PF Tasten auswendig.
Navigiert durch die Masken rauf und runter.

Kennt sich mit graphischen Oberflächen aus
Arbeitet grundsätzlich mit dem Maus.
Hat immer mehrere Fenster im Schirm offen.
Navigiert kreuz und quer durch die Webseiten.