Extracting Business Rules from Source Code

Harry M. Sneed & Katalin Erdos
SES Software-Engineering Service GmbH
Rosenheimer Landstrasse 37
D-85521 Ottobrunn/Munich, Germany

ABSTRACT

This paper reviews the state of the art on application knowledge acquisiton from existing software systems and defines the role of business rules. It then goes on to present a method for identifying and extracting business rules by means of data output identification and program stripping. This method has been implemented in a reverse engineering tool SOFT-REDOC for COBOL programs. The results are intended to aide the business analyst in comprehending legacy programs.

Keywords: Knowledge Acquisition, Business Rules, Reverse Engineering, Program Documentation

Knowledge Requirements for Software Maintenance

Work on program comprehension has been going on since the first programs were written but only recently has this work become an independent discipline closely linked to the fields of software maintenance and software reverse engineering. The purpose of the discipline is to explore means of better understanding existing computer programs so as to be able to
- maintain them in their current context,
- reuse them in another context,
- decide when to reengineer or redevelop them.

This implies that there are different requirements for comprehending programs depending on knowledge needs. The maintenance programmer who has to alter or correct a program requires structural knowledge. He needs to know the component parts of the program - the data objects processed, the interfaces, the procedures, the statements etc. - and how these parts are related to one another. The maintenance manager has to decide what to do with a program, whether to go on maintaining it as is, to reengineer it, or to redevelop it. For this, he requires information on the size, the complexity and the quality in terms of comparable numeric scales or metrics. The maintenance analyst must be able to judge how existing program can be enhanced to perform different functions or to be reused in a different context. His knowledge requirements are at the conceptual level, i.e. he needs to know what business functions a program is capable of carrying out.

Much of the early work on program comprehension has been directed towards the needs of the maintenance programmer. The classical works of Biggerstaff on design recovery [1], Rich and Wills on program design recovery [2], Hartman on program structure analysis [3] and Corbi on program understanding [4] are all representative of the work in this direction. Various techniques such as dataflow graphs, controlflow graphs, calling trees, structure diagrams and cross reference tables have been used to represent a program's architecture. There are also many commercially available tools such as VIA-INSIGHT, BATTLEMAP, REFINE, COBOL/SRE and MF-RESOLVE which aide the maintenance programmer in understanding program structures. The emphasis has moved from control structure to data structure to data flow and on to message flow and interfaces, but the objective has remained constant and that is to comprehend the effects of changes on program and data structure. Impact analysis is here the ultimate ratio.

Other work, in particular that on software metrics, has been directed towards the needs of the maintenance manager. There are a number of metrics for measuring software size such as components [5], source lines of code [6], function points [7] and data points [8]. Other metrics try to assess program complexity. Horst Zuse has identified as many as 139 different program complexity measurements [9], including those of Halstead [10], McCabe [11], Chapin [12] and Henry [13]. Finally, there are those metrics for measuring static program quality such as modularity, portability, testability, conformity etc.[14] The new ISO-9126 standard attempts to standardize them for general use. Comprehending programs through numbers has been addressed by this author in a previous paper [15]. The maintenance manager definitely needs numbers to assist him in his decision making, if not to make an objective assesment, then to at least justify his subjective opinion.

Extracting Application Knowledge from Programs

The knowledge needs of the business analyst have been addressed in a long line of papers starting with the report by Sneed and Jandrasics on the Inverse Transformation from Code to Specification in 1988 [16]. As pointed out there, the analyst must bridge the gap between the operational level at which the programs are and the conceptional level of the business process model. The objective is be able to trace segments of code to business requirements. In the same year, a paper was published by Rich and Waters on the subject of extracting higher level program representations from source code by means of abstraction and refinement [17]. The authors have demonstrated, how successive transformations can reduce the conceptual model of the program. In 1990 Karakostas introduced the teleological approach to link the software model to the object system. Teleological modelling is intended to acquire

0-8186-7283-8/96 $05.00

knowledge about the implementation, as well as the application in order to derive their interrelationships [18]. The work of Karakostas led to the development of a tool IRENE for extracting business knowledge from program source. Karakostas was one of the first to apply the concept of business rules [19].

Another more formal method for obtaining application knowledge from programs was proposed by Lano and Breuer in 1991. They suggested generating a formal specification in Z from existing COBOL programs and then analyzing the formal specification to obtain knowledge required to understand the program [20]. They later went on to implement a reverse engineering system for breaking the implementation operations down into logically coherent suboperations which correspond to components of business rules. The business rules themselves were expressed in Z++ notation [21].

A similar approach was taken by Martin Ward at the University of Durham. Ward constructed transformation algorithms for transforming source level program operations into abstract specifications using a single wide spectrum language (WSL) to represent both. Knowledge of the application domain was then filtered out of the operational details by means of successive abstractions including the introduction of abstract data variables and abstract functions [22].

The original approaches to obtaining application knowledge from source code including those published by the IBM Federal Systems Division [23], were primarily bottom up, i.e. they used only the source code as a source of information and were limited to what information could be obtained from that source. In 1993, A. Brown from Brunel challenged this approach with the contention that the idea reverse engineering could be performed in a deterministic way, building upwards from the source code through intermediate levels is as much a myth as that of forward development proceeding in a purely top-down fashion. The inverse transformation process should be an iterative process, guided by application domain knowledge, meaning the process should be viewed more as a creative process than as a mechanical procedure [24].

Layzell, Freeman and Benedusi have reinforced this approach by requesting that reverse engineering be improved by using multiple knowledge sources such as user domain knowledge, documentation and testing as well as the source itself [25]. Quilici and Chen follow the same line in their DECODE Reverse Engineering System which is query based. Queries are addressed to the system such as where is the code corresponding to a particular design element [26]. Queries are in fact requests for conceptual slices. In their paper on Legacy Code Understanding, Ning, Engberts and Kozaczynski state that queries formulated by the user isolate code segments in which statements may not be physically adjacent but are semantically related [27].

Based on this survey of the literature on knowledge acquisition through reverse engineering it would appear that the emphasis has moved from a pure mechanical transformation of code into specifications, to a more general combination of transformation techniques, using human interaction and other sources of information. It has become obvious that the knowledge needs of the business analyst can not be fulfilled by source alone. Source Program Analysis can be useful, but only if it is directed by specific knowledge requests. It is this premise upon which the research described here is based.

Concept of a Business Rule

The concept of a "business rule" was introduced in the late 1980's within the scope of IBM's AD-Cycle project. It has been defined as "a requirement on the conditions or manipulation of data expressed in terms of the business enterprise or application domain" [28] It is a verb phrase with an agent complement and a condition, e.g. to be billed after delivery the customer must have a credit rating of at least satisfactory, otherwise, the customer must pay
on delivery.

Business rules are frequently alluded to. However, the concept has remained by and large on abstract concept without an operational basis. Furthermore, there has been hardly any work on extracting business rules from real programs, mainly because the concept itself has not been adequately operationalized. Therefore, the purpose of this paper is to operationalize the concept of "business rules" and to propose a means of extracting them from business programs.

A 'business rule' at the program level is an axiom by which a business decision is made or through which a business result is calculated [29]. For instance, the decision to hire someone is a business decision. There may be several criteria for arriving at that decision, but the final outcome is a boolean yes or no. Other kinds of decisions may have more alternative outcomes, for instance price calculation or computing interest on loans. In either case there are rules for making the decision in one way or the other. The outcome of a business decision is normally reflected in a single value or a set of related values. A business rule is the function which generates these values.

In effect, business rules are a set of conditional operations attached to a given data result. Therefore, to derive business rules one must know what data they produce. It has been pointed out by others in the database area, that data is the essence of business information systems [30]. Business transactions are directed toward creating, maintaining and utilizing the company data stores in form of files, tables and databases. They are samples of programming in the large. In designing business data processing transactions, one deals with aggregate entities such as programs, modules, records and files. Business rules are

at a lower semantic level where one is dealing with elementary data items and program statements. A business rule is a function with a set of arguments and one or more results as expressed in the equation:

X = f (Y1,Y2,Y3)

Identifying Business Rules in Programs

This definition was used as a basis for deriving business rules from COBOL programs by the tool COBOL-REDOC. The key to identifying the rules is the data they use. If the user knows the outputs of a given business rule, it will be possible to determine how those results were calculated and which arguments were used. The objective is for any application relevant data result to determine which statements create or change it, where those statements are located and under what conditions they are executed. The business rule will then be composed of 4 elements=

results, arguments, assignments and conditions

in the form

<result> <== assignment
(<arguments>)
IF (<conditions>)

This may seem to be a trivial problem, after all it is the simplest form of specifying program logic. However, it is not at all trivial to recapture this simple business logic after it has been hard coded into a program. The arguments for a business rule may be derived from many different sources - some from the database, some from the user at the terminal and some from other programs. The assignment statements may be scattered throught the program. The conditions which trigger those assignments may not be at the same location as the assignments themselves. In complex programs assignments are often nested in a cascade of conditions, meaning that the assignment is at the end of a decision tree branch. The only easily locatable element in a business rule is the result, which is as a rule printed out in a report or displayed in a window. Using the result and backtracking through a program path is the only way to reassemble a business rule after the fact.

That is why in developing the tool SOFT-REDOC it was decided to approach the problem of extracting business rules from programs by using the data result as a point of entry. The analyst should know the name of the output data value, e.g. PRICE. To help him a list of all output data declared in a particular program or set of programs is presented in a scroll window. The user need only select those data items which are the result of relevant business rules. This is the means of separating real business results from all of the other technical and intermediate data (see Figure 1).

Extracting Business Rules from Programs

Once the user has identified the result of a business rule, the other elements of the rule can be obtained automatically. The SOFT-REDOC dictionary is a list of all data with pointers to their references. The references can be easily categorized as
- usage references,
- conditional references and
- assignment references.

Usage references are statements where this particular variable is used to create or alter another variable. Conditional references are statements where the state of this variable is queried or compared. Assignment references are those where this variable is created or altered. The assignment references are the key to extracting a business rule.

So the first task of the automated business rule extraction is to collect all of the assignment statements where the result in question is assigned e.g. new-price = old-price + (old-price x inflation-rate);

The same result field may have different assignments to it in different locations of the program. For instance, the price may be increased by the value added tax later on in the program

new-price = new-price + (old-price x VAT)

It is important to capture not only the assignment statement itself but also the location of each assignment, i.e. the program source name and the line number (see Figure 2).

The final outcome of this step may appear as follows

```
Line No. Assignment
101  new-price = 0;
120  new-price = old-price;
122  new-price = old-price +
       (old-price x inflation-rate);
160  new-price = new-price +
            (new-price x VAT);
```

The next step is to capture the conditions which trigger the assignments. This cannot be accomplished through the data reference list. Instead, a separate decision tree of the program as a whole is needed where the decision logic is represented in the form of a nested structure (see Figure 3). Using the line numbers of the assignments, it is possible to establish a relationship from the assignments to the conditions in the decision tree. An assignment not masked by a decision will be executed in any case such as the initialization assignment

new-price = 0;

at the beginning of the program. These assignments are unconditional.

Most assignments will, however, be conditional ones, i.e they are embedded within selection or repetition structures. The remaining price assignments could be contained in a loop such as

```
110 For each Article
 DO:<Assignments>
150 Enddo;
```

The next two assignments may be then and else branches of a condition, e.g.

```
119 IF ((inflation-rate <
        0,02) then
120 new-price = old-price;
121 else
122 new-price = old-price +
  (old-price x inflation-rate);
123 Endif
```

The final assignment may come in a later routine which calculates the tax rate

```
114 IF Article-Type = table
145 new-price = old-price +
        (new-price x VAT);
150 Endif
```

Having collected the conditions linked to the assignments the business rule can now be stated as follows:

```
For each Article DO:
new-price = 0
IF (inflation-rate < 0,02) then
 new-price = old-price;
else
 new-price = old-price + (old-price x inflation- rate);
Endif
 IF Article-Type = Taxed-Article
  new-price = old-price + (new-price x VAT)
 Endif
Enddo;
```

What has happened, is that the program has been stripped and reduced to a partial program containing only those statements which affect the outcome price. In between many other results may be produced by the same code e.g. the amount of articles on stock, a list of articles whose stock level is too low and supply orders to suppliers. It is common for most business programs, that they are producing various procedurally related results from the same code at the same time. However, in the case of a business rule, the analyst is only interested in one particular result and how this one result is calculated. The calculation of the other results is distracting and should be blinded out.

This concept of program stripping or slicing is not new. It has been explored in research time and time again [31]. Yet there are few commercial tools available to support it in regard to isolating and extracting business rules. *SOFT-REDOC* is intended to be such a tool. Not only does it extract business rules but it also generates a data dictionary with the references to each data item, a procedure tree which depicts the structure of the component parts of a program, and a decision tree which represents the conditional logic of each program. Finally, it generates a table of program interfaces, both with other programs, i.e. CALL interfaces, and with the data environment i.e. data access interfaces. However, as pointed out here, the main purpose is to extract business rules for the analyst to aide him in comprehending the business functions buried within the programs. To what extend this will really help in comprehending old programs remains to be tested, but the first evidence is positive. An example of what an extracted business rule looks like is illustrated in Figure 4.

Conclusions

The conclusion of the practical experience described here is that business rules can be extracted from source code, if certain preconditions are fulfilled. First, the variable names must be meaningful to the analyst who is working with them. Second, the analyst must know the names of the critical business output data. Third, there has to be some kind of tool support to strip the program code down to the essential elements. Fourth, there must be a technique for data flow analysis and for extracting partial paths. Finally, there must be a presentation technique to assemble and display the rules.

The key issue is that the data outputs, their operations and their predicate conditions make up the rules. If they are accessible, the rules can be derived, but only at the source statement level. To aggregate the rules into a more abstract semantic level will require a higher order language such as Z or VDL [32]. This can follow as a next step after the rules have been extracted, provided that the user really requires a higher level of abstraction.

REFERENCES

[1] Biggerstaff, T. (89): 'Design recovery for maintenance and reuse', IEEE Computer, 7, July 89, p.36-49.

[2] Rich, C./Wills, L. (90): 'Recognizing program's design - a graph-parsing approach' IEEE Software, 1, Jan. 1990, p. 82-89

[3] Hartman, J. (91): 'Under-standing natural programs using proper decomposition' in Proc. of ICSE-91, Austin,Texas, 1991, p. 62-73

[4] Corbi,T. 89): 'Program Understanding - Challege for the 1990s' IBM Systems Journal, Vol. 28, No. 2, 1989, p.294-306

[5] Wolverton, R. (74): 'The costs of developing large-scale software' IEEE Trans. on Computers, Vol. C-23, No. 6, June 1974, p. 615

[6] Boehm,B. (81): 'Software Engineering Economics', Prentice-Hall, Englewood Cliffs, N.J., 1981
[7] Albrecht,A.J./Gaffney,J.E. (83): 'Software Function, Source Lines of Code and Development Effort Prediction' IEEE Trans. on S.E., Vol. SE-9, No. 6, Nov. 1983, p.639
[8] Sneed,H. (90): 'Die Data-Point Methode' ONLINE ZfD, No. 5/90, May 1990, p. 48
[9] Zuse,H.(91): 'SoftwareComplexity Measurement' DeGruyter Verlag, Berlin 1991
[10] Halstead,M. (75): 'Elements of Software Science', Elsevier North-Holland, Amsterdam, 1975
[11] McCabe,T.J. (76): 'A Complexity Measure', IEEE Trans. on S:E, Vol. 2, No.4, 1976, p.308-320
[12] Chapin,N. (79): 'A measure of software complexity' in Proc. of National Computer Conference, Chicago, 1979, p. 58-67
[13] Henry,S./Kafura,D. (81): 'Software structure metrics based on information flow' IEEE Trans. on S.E. Vol. SE.-7, No.5, 1981, p.510-518
[14] ISO/IEC (94): 'ISO-9126 Software Product Evaluation' International Standards Organization, Geneva, 1994
[15] Sneed,H. (94): 'Program Comprehension through numbers' in Proc of 3rd Conf. on Program Comprehension, Washington,D.C, Nov. 1994
[16]Sneed,H./Jandrasics,G.(88):'Inverse transformation of software from code to specification' Proc. of CSM'88, Phoenix, Arizona, IEEE Press, New York, p. 102-109
[17] Waters,R.C. (88): 'Program Translation via Abstraction and Reimplementation' IEEE Trans. on S:E:, Vol. SE-14, No. 8, 1988, p. 1207-1228
[18] Karakostas, V. (90): 'Modelling and Maintenance of Software Systems at the teleological level' in Journal of Software Maintenance, Vol. 2, No.1, 1990, p. 47
[19] Karakostas, V. (92): 'Intelligent Search and Acquisition of Business Knowledge from Programs' in Journal of Software Maintenance, Vol. 4, No. 1, 1992, p.1
[20] Breuer,P.T./Lano,K. (91): 'Creating specifications from code - reverse engineering' in Journal of Software Maintenance, Vol. 3, No. 3, 1991, p. 145
[21] Lano,K./Breuer,P.T./Haughton, H. (93): 'Reverse-engineering COBOL via Formal Methods' in Journal of Software Maintenance, Vol.5, No.1,1993, p.13
[22] Ward,M. (93): 'Abstracting a Specification from Code' in Journal of Software Maintenance, Vol. 5, No. 2, 1993, p.101
[23] Hausler,P/Pleszkoch,M./Linger, R./Hevner,A. (90): 'Using Function Abstraction to understand program behavior' IEEE Software, Jan. 1990, p. 55-63
[24] Brown, A.J. (93): 'Specifications and Reverse engineering' in Journal of Software Maintenance, Vol. 5, No. 3, 1993, p.147
[25] Layzell,P./Freeman, M./Benedusi, P. (95): 'Improving Reverse-engineering through the Use of Multiple Knowledge Sources' in Journal of Software Maintenance, Vol. 7, No. 4, p. 279
[26] Quilici,A./Chen,D. (95): 'DECODE - A cooperative environment for Reverse- engineering legacy systems' in Proc. of 2nd Workshop on Reverse engineering, Toronto, IEEE Computer Press,1995, p. 156-165
[27] Ning,J./Engberts, A./Kozaczynski,W. (93): 'Legacy code understanding' in Comm. of ACM, Vol. 37, No. 5, 1993, p. 50-57
[28] Selfrige, P./Waters,R./ Chikowsky,E. (93): 'Challenges to the field of Reverse-engineering' in Proc. of 1st Workshop on Reverse-engineering, Baltimore, IEEE Computer Press, 1993, p. 144-151
[29] Jarzabek,S. (94): 'Life-cycle approach to strategic re-engineering of Software'in Journal of Software Maintenance, Vol. 6, No. 6, 1994, p.287
[30] Aiken,P./Muntz,A./Richards/R. (94): 'DOD Legacy Systems -Reverse-engineering data requirements'in Comm. of ACM, Vol. 37, no. 5, 1994, p. 26-41
[31] Weiser,M. (84): 'Program Slicing' IEEE Trans. on S.E., Vol. SE-10, No. 4, 1984, p. 252-357
[32] Ward,M./Bennett,K. (95): 'Formal Methods for legacy systems' in Journal of Software Maintenance, Vol. 7, No. 3, 1995, p. 221

```
---
 LEVEL TYPE    DATA NAMES (IN PROGRAM: COBOLD)     PICTURE
 ---------------------------------------------------------------
---
 0   FILE  FILE-DATA
 ---------------------------------------------------------------
---
 1   REC     CUSTOMER-RECORD
 2   AN       REC-TYPE                  PIC XX
 2   NUM      CUST-NO                   PIC 9(8)
 2   AN       CUST-NAME                 PIC X(20)
 2   STRU     CUST-ADDRESS
 3   AN        STR                    PIC X(30)
 3   NUM        ZIP                  PIC 9(4)
 3   AN        CITY                  PIC X(20)
 3   AN        STATE                  PIC X(4)
 2   NUM      CUST-CREDIT                PIC 9
 ---------------------------------------------------------------
--
 1   REC    ARTICLE-RECORD
 2   AN      REC-TYPE                  PIC XX
 2   NUM     ART-NO                    PIC 9(8)
 2   AN      ART-TYPE                  PIC X(4)
 2   NUM     ART-QUAN                    PIC S9(5)
 2   NUM       ART-PRICE                     PIC
S999V99
 2   AN      ART-NAME                  PIC X(40)
 ---------------------------------------------------------------
--
 1   REC    ORDER-RECORD
 2   AN      REC-TYPE                  PIC XX
 2   NUM     ORDER-NO                    PIC 9(8)
```

```
2   NUM     CUST-NO                  PIC 9(8)
2   STRU    ORDER-ITEMS
3   STRU     * ORDER-ITEM (3)
4   NUM        ITEM-NO               PIC 9(3)
4   NUM        ART-NO                PIC 9(8)
4   AN         ART-TYPE              PIC X(4)
4   NUM        ITEM-QUAN               PIC 9(5)
 ------------------------------------------------------------------
--
1   REC   DISPATCH-RECORD
2   AN     REC-TYPE                  PIC XX
2   NUM    ORDER-NO                    PIC 9(8)
2   NUM    CUST-NO                   PIC 9(8)
2   AN     CUST-NAME                   PIC X(20)
2   STRU   CUST-ADDRESS
3   AN      STR                   PIC X(30)
3   NUM      ZIP                  PIC 9(4)
3   AN      CITY                  PIC X(20)
3   AN      STATE                  PIC X(4)
2   STRU   ORDER-ITEMS
3   STRU     DISPATCH-ITEM
4   NUM        ITEM-NO               PIC 9(3)
4   NUM        ART-NO                PIC 9(8)
4   NUM        ART-TYPE               PIC 999
4   NUM        ITEM-QUAN               PIC 9(5)
2   AN     PAYMENT                   PIC X
 ------------------------------------------------------------------
--
1   REC   POSITION-RECORD
2   AN     REC-TYPE                  PIC XX
2   NUM    ORDER-NO                    PIC 9(8)
2   NUM    CUST-NO                   PIC 9(8)
2   STRU   ORDER-ITEMS
3   STRU     DISPATCH-ITEM
4   NUM        ITEM-NO               PIC 9(3)
4   NUM        ART-NO                PIC 9(8)
4   NUM        ART-TYPE               PIC 999
4   NUM        ITEM-QUAN               PIC 9(5)
 ------------------------------------------------------------------
--
1 AN   PRT-LINE                      PIC X(133)
 ------------------------------------------------------------------
---
```

Figure 1 : Data used by Program

```
 ------------------------------------------------------------------
--
-    DATA REFERENCED IN PROGRAM: COBOLD
-
 ------------------------------------------------------------------
--
0 FILE FILE-DATA

**************************************************
***************
1 REC  ARTICLE-RECORD
    ------------------------------------------------------------
-
 2 NUM     ART-QUAN                    PIC S9(5)
  272 R        SUBTRACT ITEM-QUAN IN ORDER-
RECORD (POS) FROM
                ART-QUAN IN ARTICLE-RECORD
    ------------------------------------------------------------
-
 2 NUM      ART-PRICE                        PIC
S999V99

**************************************************
***************
 1 REC  DISPATCH-RECORD
    ------------------------------------------------------------
-
 2 AN        PAYMENT                 PIC X
  313 R     MOVE "N" TO PAYMENT
  315 R     MOVE "C" TO PAYMENT

**************************************************
***************
 0 WORK  WORK-DATA

**************************************************
***************
 1 NUM  ERROR-TYPE                      PIC 99.
  247 R   MOVE ZERO TO ERROR-TYPE.
   250 R         INVALID KEY MOVE 1 TO ERROR-
TYPE
  259 R   MOVE ZERO TO ERROR-TYPE.
   263 R         INVALID KEY MOVE 2 TO ERROR-
TYPE
  269 R            MOVE 3 TO ERROR-TYPE
    ------------------------------------------------------------
-
 1 NUM  PRICE                          PIC 99999V99
   335 R     MULTIPLY ART-PRICE IN ARTICLE-
RECORD BY ITEM-QUAN
                IN ORDER-RECORD (POS) GIVING
PRICE.
    ------------------------------------------------------------
-
 1 NUM  TOTAL-CUST-PRICE                    PIC
99999V99
  254 R   MOVE ZERO TO TOTAL-CUST-PRICE.
  338 R   ADD PRICE TO TOTAL-CUST-PRICE.
    ------------------------------------------------------------
-
   1       NUM           TOTAL-ITEMS-FULFILLED
PIC 99
   253  R        MOVE  ZERO  TO  TOTAL-ITEMS-
FULFILLED
  303 R   ADD 1 TO TOTAL-ITEMS-FULFILLED.

**************************************************
***************
 1 REC  DATA-PRT-LINE
    ------------------------------------------------------------
-
```

```
 2 EDIT     ART-PRICE                    PIC
ZZZZ9.99
 333 R   MOVE ART-PRICE IN ARTICLE-RECORD
TO
          ART-PRICE IN DATA-PRT-LINE.
     ------------------------------------------------------------
-
 2 EDIT     TOTAL-PRICE                  PIC
ZZZZ9.99
 337 R   MOVE PRICE TO TOTAL-PRICE.

**********************************************
***************
 1 REC  ERROR-PRT-LINE
     ------------------------------------------------------------
-
 2 AN       ERROR-MESSAGE                PIC
X(40)
 369 R      MOVE "CUSTOMER NOT KNOWN " TO
ERROR-MESSAGE.
 371 R      MOVE "ARTICLE NOT AVAILABLE "
TO ERROR-MESSAGE.
 376 R      MOVE "INSUFFICIENT QUANTITY ON
STOCK" TO ERROR-MESSAGE.

**********************************************
***************
```

Figure 2 : Data References

```
------------------------------------------------------------
-----
Line Level    Decision Statement          in Program:
COBOLD
------------------------------------------------------------
----
  221 0 COBOLD
  221 1   COBOLD-Code-Sequence-0000
  224 0 INITIALIZATION
  225 1   INITIALIZATION-Code-Sequence-0001
  238 0 READ-ORDERS
  239 1   READ ORDER-FILE
  240 1   AT END GO TO TERMINATION
    241 1       NOT AT END PERFORM READ-
CUSTOMER
  240 2     AT END GO TO TERMINATION
    241 2         NOT AT END PERFORM READ-
CUSTOMER
  242 2     READ-ORDERS-Code-Sequence-0004
  243 1   END-CONDITION
  243 1   READ-ORDERS-Code-Sequence-0005
  246 0 READ-CUSTOMER
  247 1   READ-CUSTOMER-Code-Sequence-0006
  249 1   READ CUSTOMER-FILE
  250 1   INVALID KEY MOVE 1 TO ERROR-TYPE
  251 2     READ-CUSTOMER-Code-Sequence-0007
  252 2     GO TO REPORT-ERROR.
  252 1   END-CONDITION
  252 1   READ-CUSTOMER-Code-Sequence-0008
>> 253   R       MOVE ZERO TO TOTAL-ITEMS-
FULFILLED
>> 254 R    MOVE ZERO TO TOTAL-CUST-PRICE.
  258 0 PROCESS-ORDER
  259 1   PROCESS-ORDER-Code-Sequence-0009
  260 1   PERFORM VARYING POS FROM 1 BY 1
UNTIL POS > 3
  261 1   OR ITEM-NO IN ORDER-RECORD (POS)
= 9
  262 2     PROCESS-ORDER-Code-Sequence-0010
  260 2     READ ARTICLE-FILE
  263 2       INVALID KEY MOVE 2 TO ERROR-
TYPE
  264 3           PROCESS-ORDER-Code-Sequence-
0011
  265 2     END-CONDITION
  265 2     PROCESS-ORDER-Code-Sequence-0012
  267 2       IF ITEM-QUAN IN ORDER-RECORD
(POS) >
  268 2     ART-QUAN IN ARTICLE-RECORD
  269 3           PROCESS-ORDER-Code-Sequence-
0013
  272 2     ELSE
  272 3           PROCESS-ORDER-Code-Sequence-
0014
>> 272   R             SUBTRACT ITEM-QUAN IN
ORDER-RECORD (POS) FROM
>>                        ART-QUAN IN ARTICLE-
RECORD
  276 2     END-CONDITION
  276 2     PROCESS-ORDER-Code-Sequence-0015
  281 0 TERMINATION
  282 1   TERMINATION-Code-Sequence-0016
  292 0 WRITE-OPEN-POSITIONS
     293    1        WRITE-OPEN-POSITIONS-Code-
Sequence-0017
  302 0 WRITE-DISPATCH
  303 1   WRITE-DISPATCH-Code-Sequence-0018
>> 303 R    ADD 1 TO TOTAL-ITEMS-FULFILLED.
  312 1   IF CUST-CREDIT > 5
  313 2     WRITE-DISPATCH-Code-Sequence-0019
>> 313 R      MOVE "N" TO PAYMENT
  315 1   ELSE
  315 2     WRITE-DISPATCH-Code-Sequence-0020
>> 315 R      MOVE "C" TO PAYMENT
  316 1   END-CONDITION
  316 1   WRITE-DISPATCH-Code-Sequence-0021
  321 0 REPORT-ORDER
  322 1   IF ERROR-TYPE > 0
  323 2     REPORT-ORDER-Code-Sequence-0023
  324 1   END-CONDITION
>> 333   R       MOVE ART-PRICE IN ARTICLE-
RECORD TO
>>            ART-PRICE IN DATA-PRT-LINE.
>> 335 R    MULTIPLY ART-PRICE IN ARTICLE-
RECORD BY ITEM-QUAN
```

```
>>                IN ORDER-RECORD (POS) GIVING
PRICE.
>> 337 R   MOVE PRICE TO TOTAL-PRICE.
>> 338 R   ADD PRICE TO TOTAL-CUST-PRICE.
------------------------------------------------------------------
----
```

Figure 3 : Program Decision Logic

```
 BUSINESS RULE FOR ART-QUAN IN ARTICLE-
RECORD IN PROGRAM: COBOLD
 ------------------------------------------------------------------
--

 1 REC  ARTICLE-RECORD
 2 NUM      ART-QUAN                    PIC S9(5)
   ------------------------------------------------------
   258 0 PROCESS-ORDER
   259 1   PROCESS-ORDER-Code-Sequence-0009
   260 1   PERFORM VARYING POS FROM 1 BY 1
UNTIL POS > 3
   261 1   OR ITEM-NO IN ORDER-RECORD (POS)
= 9
   262 2     PROCESS-ORDER-Code-Sequence-0010
   263 2     READ ARTICLE-FILE
    263 2       INVALID KEY MOVE 2 TO ERROR-
TYPE
    264 3           PROCESS-ORDER-Code-Sequence-
0011
   265 2     END-CONDITION
   265 2     PROCESS-ORDER-Code-Sequence-0012
    267 2       IF ITEM-QUAN IN ORDER-RECORD
(POS) >
   268 2     ART-QUAN IN ARTICLE-RECORD
    269 3           PROCESS-ORDER-Code-Sequence-
0013
   272 2     ELSE
    272 3           PROCESS-ORDER-Code-Sequence-
0014
 >> 272 R              SUBTRACT ITEM-QUAN IN
ORDER-RECORD (POS) FROM
 >>                ART-QUAN IN ARTICLE-RECORD

**************************************************
**********************
 BUSINESS RULE FOR PAYMENT IN DISPATCHE-
RECORD IN PROGRAM: COBOLD
 ------------------------------------------------------------------
--
 1 REC  DISPATCH-RECORD
 2 AN       PAYMENT                     PIC X
   ------------------------------------------------------
   302 0 WRITE-DISPATCH
   303 1   WRITE-DISPATCH-Code-Sequence-0018
   312 1   IF CUST-CREDIT > 5
   313 2     WRITE-DISPATCH-Code-Sequence-0019
>> 313 R     MOVE "N" TO PAYMENT
   315 1   ELSE
   315 2     WRITE-DISPATCH-Code-Sequence-0020
>> 315 R     MOVE "C" TO PAYMENT

**************************************************
**********************
 0 WORK  WORK-DATA

**************************************************
**********************
 BUSINESS RULE FOR PRICE IN WORK-DATA IN
PROGRAM: COBOLD
 ------------------------------------------------------------------
--
 1 NUM  PRICE                       PIC 99999V99
   ------------------------------------------------------
   321 0 REPORT-ORDER
   322 1   IF ERROR-TYPE > 0
   323 2     REPORT-ORDER-Code-Sequence-0023
   324 1   END-CONDITION
    335 R     MULTIPLY ART-PRICE IN ARTICLE-
RECORD BY ITEM-QUAN
                IN ORDER-RECORD (POS) GIVING
PRICE.

**************************************************
**********************
  BUSINESS RULE FOR ERROR-MESSAGE IN
ERROR-PRT-LINE IN PROGRAM: COBOLD
 ------------------------------------------------------------------
--
 1 REC  ERROR-PRT-LINE
 2 AN       ERROR-MESSAGE                PIC
X(40)
   ------------------------------------------------------
   353 0 REPORT-ERROR
   354 1   REPORT-ERROR-Code-Sequence-0026
   358 1   IF ERROR-TYPE > 1
   359 2     REPORT-ERROR-Code-Sequence-0027
   365 1   END-CONDITION
   368 1   IF ERROR-TYPE = 1
>> 369 R       MOVE "CUSTOMER NOT KNOWN "
TO ERROR-MESSAGE.
   ------------------------------------------------------
   370 2     IF ERROR-TYPE = 2
>> 371 R       MOVE "ARTICLE NOT AVAILABLE
" TO ERROR-MESSAGE.
   ------------------------------------------------------
   375 1   IF ERROR-TYPE = 3
>> 376 R       MOVE "INSUFFICIENT QUANTITY
ON STOCK" TO ERROR-MESSAGE.

**************************************************
****************
```

Figure 4 : Business Rules at the Source Level