Improving the Quality of Legacy Code by Reverse Engineering

Rosângela D. Penteado[1]
DC, Federal University of São Carlos
CP 676 – São Carlos - SP – Brazil
rosangel@dc.ufscar.br

Rosana T. V. Braga
DI, University of Franca/SCE-ICMC, University of São Paulo
CP 668 - São Carlos –SP –Brazil
rtvb@icmc.sc.usp.br

Paulo C. Masiero[2]
SCE-ICMC-University of São Paulo
CP 668 - São Carlos – SP – Brazil
masiero@icmc.sc.usp.br

ABSTRACT

An approach for improving the quality of legacy code is presented. It consists of the reengineering of systems developed with procedure orientation to object orientation based on Fusion/RE. Fusion/RE is an overall process for conducting object oriented reverse engineering in procedure oriented legacy systems. The reengineering involves no change in functionality or programming language. The proposal is an alternative to existing methods for doing reengineering directly from the legacy code, which present difficulties that are overcome by first doing the reverse engineering. An experimental application of the proposed approach to a real system with 20K lines of Clipper code is reported. Through this, it was possible to observe real improvement in the maintainability of the legacy system. Corrective, evolutionary and perfective maintenance is easier to conduct after the reengineering.

Keywords: Reuse, Maintainability, Object Oriented Reverse Engineering, Reengineering, Fusion/RE, Object-Oriented, Segmentation.

1. INTRODUCTION

Reengineering legacy information systems is of extreme importance as documentation can be lost, system development and maintenance teams can be substituted and administration rules can be changed. The systems, however, are usually kept running as the activity they support cannot be halted.
Several authors have been interested in the recycling of systems towards object orientation [5, 6, 12]. Sneed [12] shows a process for reengineering procedure oriented COBOL programs to object orientation. This process is done in two stages, the first preserving the original language, not object oriented, in which the system was developed, and the second changing the language to one version of object oriented COBOL. He emphasizes several reasons for the application of this process, among them the best use made of the features offered by the object oriented technology, the reduction of maintenance costs and the increase of programs and data reuse in the new architecture of the system. The first stage, in which code segmentation is done, is very attractive as it permits to obtain several advantages of the object-oriented approach even not doing the second.

Sneed enumerates some obstacles present in this process. First, the difficulty to separate the legacy code into the prospective methods, as they are intertwined in the code. Second, the short size of some of these methods and their high number. Third, the difficulty of establishing automatic rules for definition of the objects, because these only can be defined in the context of their use. Fourth, the difficulty to eliminate redundant code, as it is very hard to know what is redundant. Finally, the difficulty in eliminating equal and opposite names, because the legacy system uses the same name for different things. It is important to notice that Sneed conducts the reengineering without first doing the reverse engineering of the legacy system. In the same Conference where he has shown his work [12], part of the authors of this paper have proposed an approach to this problem, Fusion/RE [9], to do reverse engineering in legacy code based on the Fusion method [4].

The purpose of this paper is first to apply Fusion/RE to legacy information systems and then to do the first phase of the reengineering as suggested by Sneed, preserving the functionality, the programming language and other factors, but

[1] Financial Support from RHAE/CNPq grant 610623/95-8
[2] Financial Support from CNPq and FAPESP

changing the paradigm, from procedure to object orientation. Differently from Sneed, we do not proceed doing the change of language to one object oriented, as that author reports no problems in this phase of the reengineering. Also, because part of the authors of this paper is participating in an experiment for this phase, based on a software tool called Draco-Puc [10].

This paper shows briefly, in section 2, an introduction to the proposed approach and in section 3 this approach is applied to a sample information system. In section 4 the reengineering done is evaluated and in section 5 the conclusions are shown.

2. REENGINEERING BASED ON FUSION/RE

Figure 1 shows the scheme of the proposed approach, where we can notice that two steps are added to Fusion/RE [9], corresponding to the system reengineering. The four initial steps, referring to reverse engineering, that are briefly described here, can be found in more detail in [8,9].

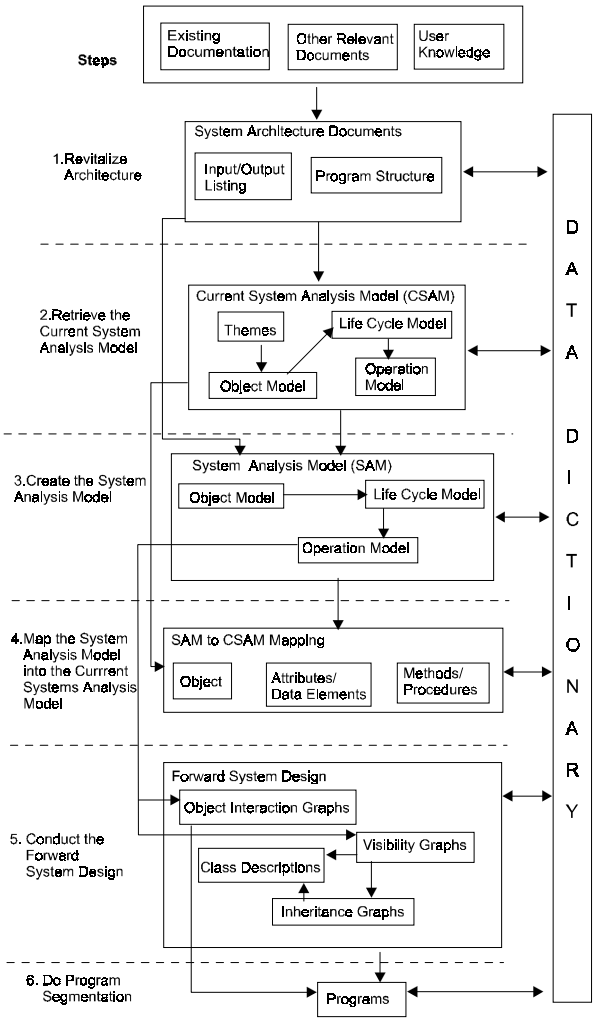

Figure 1 - Scheme of the Proposed Approach

Step one of the approach consists in revitalizing the system architecture based on documentation available, if it exists, or on the source code. Benedusi [1] and Wong [13] show tools to extract documentation from the source code, but those have not been used. Other authors [2, 5, 11], give guidelines to manually retrieve information from code, in order to help in object identification. At the end of the step, we have a list of all procedures, with their description and call hierarchy (call/called by).

Step two is concerned with retrieving the Current System Analysis Model (CSAM). In it we produce a pseudo Object Model of the system, identifying the classes and their relationships together with their attributes and associated procedures. We observe that many of these procedures contain anomalies, that is, a same procedure deals with more than one class. Those procedures are classified as (c) for constructor, when the procedure alters a data structure and (o) for observer, when the procedure only consults the data structure. When one procedure consults and alters the same data structure it is classified only as (c). The signal + associated to annotation (c) and/or (o), represents that the procedure constructs and/or observes two or more data structures. This way, the procedures with anomalies can be classified as (oc), (o+c), (oc+) and (o+c+). The procedures are classified as (i) when they refer to the implementation type utilized. Next, we define the Life Cycle Model, which portrays the system global behavior, concerning the chronological order in which the operations are performed. These are the bases to construct a system Operation Model, in which each operation is described in detail, according to a pre-established template. In fact, it is a pseudo object oriented model, as it does not attend all the restrictions of object orientation.

Step three is concerned with creating the System Analysis Model (SAM), focusing the application domain and not the implementation. The object model, the life cycle model and the operations model are abstracted from those constructed previously, with some additional care, as for example, to change the names to gain expressiveness. The procedures without anomalies generate methods directly, whereas those with anomalies have to be divided in several methods, so that each one is associated with only one class. Another aspect of the analysis is the preparation of a scenario. The system inputs and outputs need agents to control them and we must identify these agents through system functions, creating the corresponding scenarios. A scenario is a sequence of events that run between agents and the system with some purpose. Each scenario involves agents, the tasks that they want the system to do and the sequence of communications involved in those tasks. The scenarios help in the preparation of the life cycle and operation models.

Step four does a mapping from the System Analysis Model to the Current System Analysis Model, describing the relationships between them. The classes, attributes and methods of the SAM are mapped into the corresponding elements of the CSAM, with annotation of possible inclusions/exclusions. This step is important in future maintenance and reuse of the system, in case only reverse engineering is done.

Step five takes care of the forward system design, considering the reengineering with change of the implementation paradigm, but without changing the functionality. In this step we construct the Object Interaction Graphs, Visibility Graphs, Class Description and Inheritance Graphs, according to the Fusion Method [4].

The Object Interaction Graphs are constructed based on the Operation Model. They show the communication between objects, by methods, distinguishing operation controller classes from their collaborator classes. A pseudocode is written at the operation method controller class showing its logic and the sequence of calling the methods of the collaborator classes. It is important to notice that the method of the controller class (that implements an operation) interacts with more than one object, but this occurs through the methods of the collaborator classes, so avoiding anomalies to occur.

The Visibility Graphs establish the communication between classes, limiting which can exchange messages and of which types. In the Object Interaction Graph it is assumed that all objects are mutually visible, and can exchange messages among themselves, but this is restricted in the visibility graph.

The Class Descriptions contain complementary documentation to the system object model. To each class is joined information that is distributed among the elements of the documentation now produced. From the Object Interaction Graph the methods and parameters are extracted; from the Object Model some data attributes are extracted and from the Visibility Graphs the object attributes are extracted. For each class its entire interaction with other classes is specified.

The Inheritance Graphs are a mechanism through which a class can be defined as a specialization of another.

The procedures with anomalies can be examined in face of the documentation produced in step five. The result of this exam will dictate how to perform the next step. In that step the original program can be split into methods. Tests can give evidences that the same situation is executed in the same order.

Step six treats the system segmentation. The original procedures with anomalies are here divided in methods defined in the previous step. This definition of the methods becomes a strong guide for each line of code to be joined with the method that it belongs to. Thus, the difficulties in doing the segmentation, according to Sneed [12], are attenuated.

The code is examined and comments are inserted establishing a correspondence between this code and the methods that were defined. The pseudocode of the methods of the controller classes is reviewed to consider implementation details. This done, the parts corresponding to each method are put together.

Two key points should be noticed by the reader, to understand how steps 5 and 6 work. First, the segmentation of the program is done considering the forward system design, developed according to the Fusion design step, that was conducted as if the system would be re-programmed using an object oriented language. This facilitates the segmentation, as was already observed, and eliminates the difficulties shown by Sneed. Second, the segmentation does not change the programming language, which is not object-oriented. It is a code programmed with the style of abstract data types and in which the mechanisms of specialization/generalization and aggregation can be simulated [7]. However, after the segmentation, it can be easily translated to an object oriented language as Sneed has done with OO COBOL [12].

We have observed that, in the initial process, the submodule that interacts with a specific class continues to have activity with something else, as for example: interface, physical aspects of the system, etc. These activities must be isolated in other submodules. After that we can proceed doing the test of the system re-segmented, to assure that it has the same functionality.

3. THE CASE OF REENGINEERING A CLIPPER SYSTEM

An experiment was done with the process presented in section 2 to assess its application to an information system developed in Clipper, with 20.000 lines of code. The system takes care of the services done in a mechanic and electric car repair shop, also doing stock control of the parts used. The customer goes to the shop to ask for services in his vehicle. A customer can have several vehicles. The same vehicle can return to the shop many times; a distinct customer order is prepared in each visit. This contains data about the customer, the vehicle and the services to be done. When the service is completed, the customer order is fulfilled with the parts used and the labor work performed. Sometimes the repair can need parts that do not exist in stock. These are acquired outside and have to be included in the customer's order. The recording of these parts is important to the shop management because they are candidates to be stocked. The possible vehicle models need to be recorded in the system, so that price tables are utilized for payment of services, both electrical and mechanical, according to the vehicle model.

We have chosen this system because it is a real working system, presents maintenance difficulties and is obsolete in relation to the user interface. The six steps of the proposed approach were applied to this system, as described in the following paragraphs.

In step 1, as the documentation of the system was considered reasonable, it was necessary only its re-organization. There was a detailed description of the relational tables of the database, of the system menus and, for each procedure, the "call/called by" relationship.

In step 2, the Current System Object Model was constructed, based on the analysis of the databases (files with extension .DBF) and of the programs (files with extension .PRG). We selected as classes all files that contain information on the system. In fact, this is a pseudo object model, as it is the result of an adaptation of the procedural implementation to an object oriented one. In the left side of Figure 2, some pseudo classes of the actual implementation are presented. In the right side, we show the procedures of the legacy system, as for example the module Coscree1, responsible to entry a customer order. It contains anomalies, as it interacts with the pseudo classes: Customer, Custord, Custvehi and Vehitype. We can observe the anomalies by the four arrows going from this procedure to different pseudo classes. Figure 3 shows, partially, the system Life Cycle Model. The system Operation Model was also developed, but is not shown here due to space limitations.

In step 3, the pseudo classes of CSAM were abstracted to classes of SAM. Following the guidelines contained in [3], this abstraction was done in seven stages. In the first stage the foreign keys of the tables were represented in the model as relationships. In the second, the tables implementing

relationships were modeled as relationships, and the corresponding keys were inserted in the appropriate classes. In the third, class aggregation and specialization were identified. In the fourth, system inputs and the external agents that originate them were identified. In the fifth, system outputs and the corresponding external agents were identified. In the sixth, name generalization for the model produced in the fourth stage and class attributes have been added. In the seventh, name generalization for the model produced in the fifth stage was done. The models obtained in the two last stages constitute the abstracted object model. In the left side of figure 4 some classes abstracted from CSAM are shown. In the right side the methods obtained after eliminating the anomalies of the procedure "Coscree1" of figure 2 are shown.

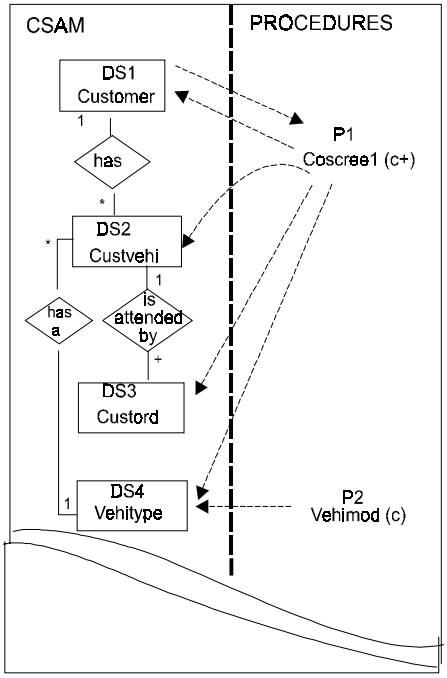

Figure 2 - CSAM and Legacy Code Procedures

In step 4, the mapping between SAM and CSAM classes was easily produced, as the detailed transition described in the previous step provided no loss of information. This mapping shows clearly what each of the methods does. So the system maintenance is eased because we know exactly which piece of code must be altered or modified.

In step 5, the Object Interaction Graph, shown in figure 5, was prepared, based on the Operation Model and on the Life Cycle Model. As we can observe, Customer Order is the controller class of this operation. The Class Descriptions and the Visibility Graph were also developed, but are not included here due to space shortage.

In step 6, the legacy code was divided in methods, according to the anomalies identified in step 2 and eliminated in step 3. In our experiment we could observe the points alerted in [12]: there were several methods and almost all of them contained just a few lines of code. But the possibility of reusing them has been increased because they have well defined functionality.

Lifecycle Electrical and Mechanical Car Repair Shop : =
(Transaction | File_Update | Print | Search)*

Transaction = (entry_customer_order . [#CO_printed]) |
modify_customer_order | eliminate_customer_order | . . .)

File_Update = (insert_part | insert_customer | . . .)

Print = (pending_CO_Report | monthly_CO_Report | . . .)

Search = (search_part | search_customer_order | ...)

Figure 3 – Partial system Lifecycle Model

Figure 6 shows a sample of the legacy code and its corresponding segmented code. Figures 7 shows methods of the segmented code. We observe that the program structure and functionality remain unaltered, as the pieces that contain anomalies are isolated in modules (in truth, methods). For example, the command "DO OBTNEXCO" refers to the method shown in figure 7, that obtains the next customer order available. Due to Clipper limitations it was necessary to abbreviate the method names of SAM to a maximum of eight characters, for example, OBTNEXCO, showed in Figure 7, corresponds to method Obtain_Next_CO-Num.

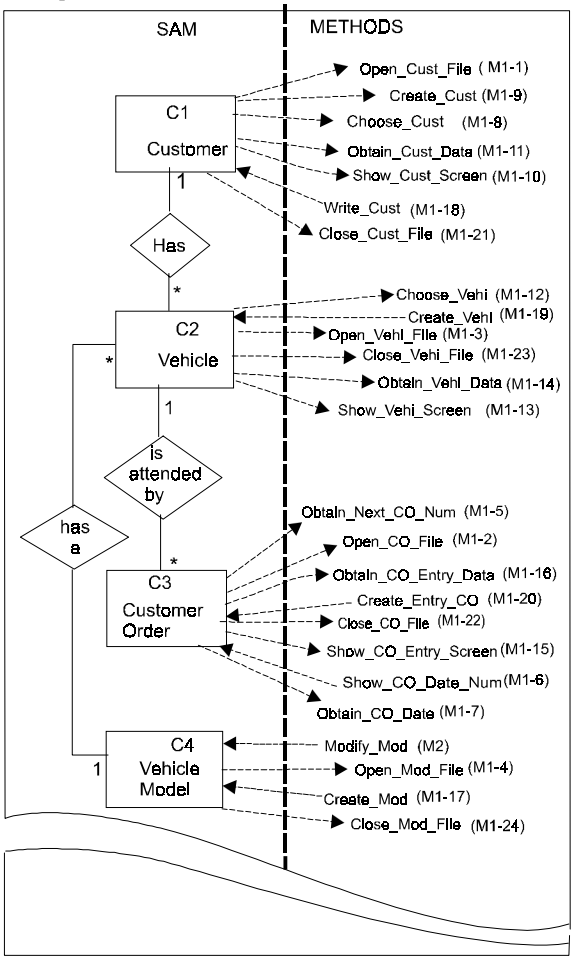

Figure 4 – SAM and corresponding methods

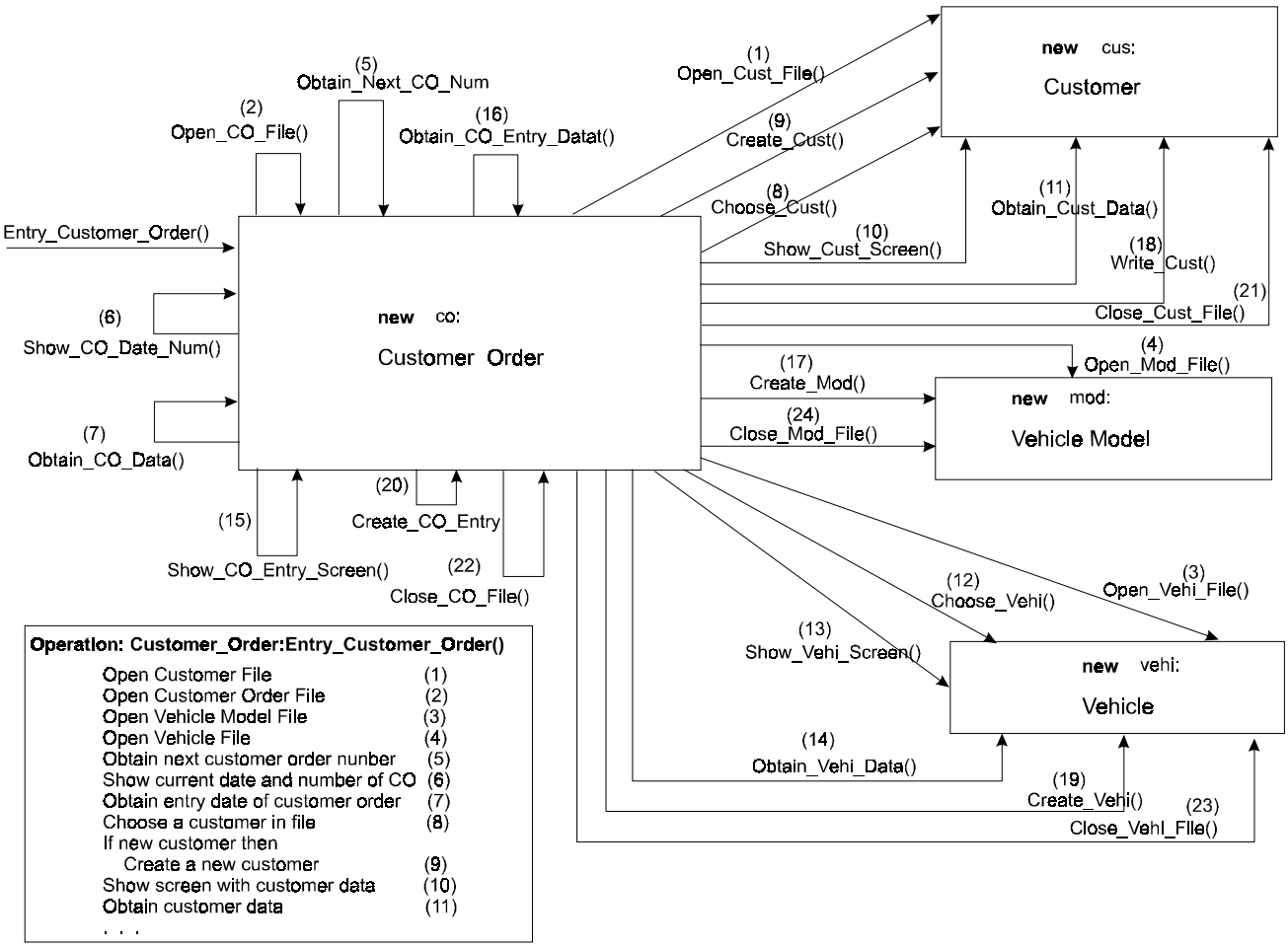

Figure 5 - Example of the System Object Interaction Graph

As it can be observed in figure 5, Customer Order is the controller class of the operation "Entry_Customer_Order". This class continues interacting with the others by message exchange that, in the case of Clipper, is implemented by the calling of procedures/functions.

It is important to notice that, after the segmentation, extentions and maintenance can be performed more easily due to the object-oriented paradigm.

4. EVALUATING THE REENGINEERING PERFORMED

As it has been seen in section 3, applying the approach described in section 2, we obtain a configuration of the legacy system similar to that produced if the first phase of the procedure proposed by Sneed was applied. Our path is longer than that suggested by Sneed, but overcomes a big piece of the difficulties that he identifies when his approach is applied.

For example, the difficulty in splitting the code into the prospective methods is attenuated as, when we reach the point in which this has to be done, we know beforehand for each procedure which methods have to be generated. Thus it is less confuse to know to what method a piece of code belongs to and through segmentation its separation is done. This guideline helps disentangle the code when it is intertwined.

Thus, the code produced in step 6 is supported by the design documentation done in step five. With this, we obtain a more secure support to change other factors of the implementation in future reengineering as, for example, the programming language. Compared with the approach proposed by Sneed, the code segmented appropriately, with the documentation generated, enhances the possibility and the facility of new reengineering. We can think about the methods of the operation controller classes as structures that facilitate the reuse of the methods of the collaborator classes. Actually, they form building blocks to increase and enlarge the system.

Martin and Odell have discussed in their book [7] an approach that can be followed to implement an object oriented system using a procedural language like Cobol, Fortran, etc. In addition to what we have done, provisions should be made to simulate inheritance, specialization, aggregation, etc. to conform to the approach suggested by them.

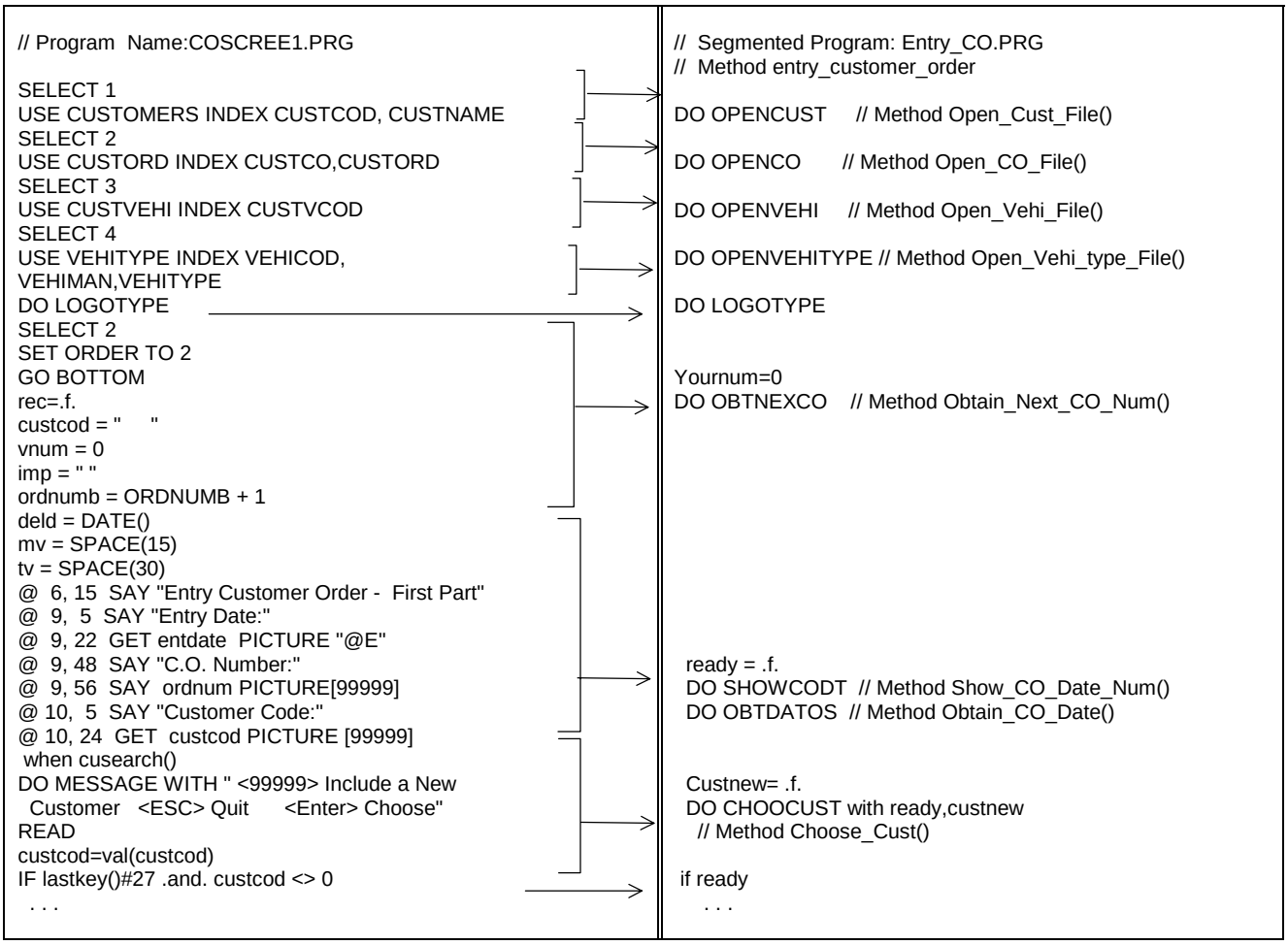

Figure 6 - Piece of Clipper Program with procedure orientation (in the left) and its corresponding piece of segmented Clipper program (in the right)

```
FUNCTION OBTNEXCO()   // Method Obtain_Next_CO_Num()
* method of Customer Order to obtain the next customer order
* number available

  SELECT CUSTORD
  SET ORDER TO 2
  Goto bottom
  Yournum = CONUM + 1
  SET ORDER TO 1

RETURN NIL
```

(a)

```
FUNCTION SHOWCODT()   // Method  Show_CO_Date_Num()
* method of Customer Order to show customer order date and
* number

@  6, 15  SAY "Entry Customer Order - First Part"
@  9,  5  SAY "Entry Date:"
@  9, 48  SAY "C.O. Number:"
@  9, 56  SAY  Yournum PICTURE[99999]
@ 10,  5  SAY "Customer Code:"

RETURN NIL
```

(b)

```
FUNCTION CHOOCUST()   //  Method Choose_Cust()
* method of Customer to choose a customer from list

PARAMETERS Ready, NewCustomer

Custcod=0
@ 10, 24  GET  custcod PICTURE [99999]  when cusearch()
DO MESSAGE WITH " <99999> Include a New Customer
<ESC> Quit      <Enter> Choose"
READ
IF lastkey()=27 .or. custcod=0
  Ready = .f.  // a customer was not choosen
ELSE
  Ready = .t.
  IF custcod=99999
    NewCustomer = .t.  // a new customer will be inserted
  ENDIF
ENDIF

RETURN NIL
```

(c)

Figure 7 - Clipper program corresponding to methods of the system

We had the concern to prepare the contents of the figures in an articulate way, so it is worth to revisit them in more detail. In figure 6 left, a piece of the legacy code is reproduced. Observing figure 2 together with it we can see that the code refers to procedure Coscree1. Its interaction with different objects can be observed by its opening of different tables: Customer, Custvehi, Custord and Vehitype. This constitutes the anomaly noted by c+ in figure 2. In figure 4 the methods created to avoid this anomaly are shown. In figure 5 we see that most of these methods belong to collaborator classes of this operation, as for example method Choose_Cust() of class Customer. In the right side of figure 6 these methods are called in the piece of segmented code of the operation "entry_customer_order". In figure 7 the code of three methods showed in figures 4, 5 and 6 is exhibited.

One advantage of our approach is that the segmentation can be conducted with the in house team, without additional training in a new object oriented programming language. The reverse engineering part of the process demands a good deal of training if also done with in house staff. Combining internal staff with an external consultant expert in object oriented development would be a feasible solution.

5. CONCLUSIONS

An approach to do reverse engineering followed by a reengineering, with the purpose of changing the orientation paradigm of the system, initially procedurally oriented, to object oriented, without change of functionality or programming language, was presented.

This approach presents the advantage of producing better compatibility between the analysis model of the object oriented system done by Fusion/RE and the legacy code. If the reengineering here discussed were not done, we would have more difficulty in recognizing in the code the corresponding components of the analysis model. The resulting segmented code is easier to understand, maintain and reuse.

Within the reengineering done, this is recognized immediately, even more because it has all the documentation of the design "a la" Fusion produced by the forward engineering embedded in the reengineering procedure. To continue the evolution of the system, as for example in the reengineering with change of functionality, this compatibility of the code with the analysis model is of utmost importance.

We have further observed that in the initial procedures segmented there was an increase in the lines of code number of about 80%. However, as the work proceeded, there was a reduction of about 40% thanks to methods reuse. The larger possibility of reuse in other systems, due to better defined methods functionality, should be considered when looking at these numbers.

6. ACKNOWLEDGMENTS

The authors thank Dr. Fernão Germano for his active participation in the research project this paper refers to and Tiemi C. Sakata for the application of the proposed approach to the actual system described.

REFERENCES

[1] Benedusi, P.; Cimitile, A.; De Carlini, U., Reverse Engineering Processes, Design Document Production, and Structure Charts. The Journal of Systems and Software, Vol. 19, No. 3, pp. 225-232, 1992.

[2] Biggerstaff, T. J.; Mitbander, B. G.; Webster,D. E., Program Understanding and the Concept Assignment Problem. Communications of the ACM, Vol. 37, nº5, 1994.

[3] Braga, R. T. V.; Masiero, P. C., Detailing of Fusion/RE Abstraction Analysis Model Step. Technical Report number 70, ICMC-USP, São Carlos-SP,1998 (In Portuguese).

[4] Coleman D. et al, Object Oriented Development - The Fusion Method. Prentice Hall, 1994.

[5] Gall, H.; Klösch, R., Finding Objects in Procedural Programs: An alternative Approach. Proceedings of the 2nd Working Conference on Reverse Engineering, Toronto, Ontario, Canada. IEEE, pp. 208-216, 1995.

[6] Jacobson, I.; Lindström, F., Re-engineering of Old Systems to an Object-Oriented Architecture. Proceedings of the OOPSLA'91, ACM, pp. 340-350, 1991.

[7] Martin, J.; Odell, J., Object-Oriented Analysis and Design, Prentice Hall, 1992.

[8] Penteado, R.D., A Method for Object Oriented Reverse Engineering. ScD Thesis, USP, Institute de Física de São Carlos, 1996.(In Portuguese)

[9] Penteado, R.D., Germano, F.; Masiero, P.C., An Overall Process Based on Fusion to Reverse Engineer Legacy Code. III Working Conference on Reverse Engineering, IEEE, Monterey, California, pp. 179-188, 1996.

[10] Prado, A.F., Domain Oriented Software Reengineering Strategy. ScD Thesis, PUC-RJ, 1992. (In Portuguese)

[11] Sneed, H. M.; Nváry, E., Extracting Object-Oriented Specification from Procedurally Oriented Programs. II Working Conference on Reverse Engineering, IEEE, 1995.

[12] Sneed, H. M., Object-Oriented COBOL Recycling. III Working Conference on Reverse Engineering. IEEE, Monterey, California , pp. 169-178, 1996.

[13] Wong, K. et al, Structural Redocumentation: A Case Study. IEEE Software, Vol.12, No.1, pp. 46-54, 1995.