Recovering Conceptual Data Models is Human-Intensive

F.Abbattista, F.Lanubile, and G.Visaggio

Dipartimento di Informatica, University of Bari
Via Amendola 173, 70126 Bari, Italy
fax: +39-80-243196 - email:giuvis@vm.csata.it

Abstract[1]

To handle the complexity of modern software systems, a software comprehension strategy pointing out the conceptual abstraction level is necessary. In this context, the role of technology is only marginal as the latter produces results which are too closely linked to the implementation aspect, whereas the conceptualization task is typically a human activity.

This study proposes a method for data reverse engineering, which, although founded on previous experiences reported in the literature, integrates them with an intensive use of human intervention, thus making it possible to bridge the gap between the implementation of solutions and the conceptual requirements of applications. The reverse engineering of data is guided by the expectations provided by a reference conceptual model, which captures the essence of the application domain, ignoring any specialist aspects stemming from particular technological and organizational solutions.

The method was experimented by reconstructing, in a banking application, the conceptual model of data and the data dictionary. The lessons learned enabled us to conclude that even if the productivity measured was relatively low, the method attains the target of recovering working software systems. Furthermore, the experimentation showed that the effectiveness of the data reverse engineering process tends to increase as time goes on and experience accumulates.

1. Introduction

Over the last twenty years, organizations which have invested in information technology have amassed enormous assets in the way of operative software. Bachman in [1] calculated that the total number of lines of Cobol code on IBM computers amounted to 77 billion. A wide knowledge of the application domain and great direct experience of user requirements have accumulated in the existing applications, with the stratification of modifications. However, many changes have been inserted haphazardly without any rationale and even when changes are based on precise design decisions, these have not been sufficiently documented. This has resulted in working software assets with a wide base of experience but which are difficult to maintain and have low potential for reuse.

To overcome these problems, reverse engineering has been subject to an extensive research, for example [3], [4], [6], [7], [8], [10], [12], [14], [15], [17]. The aim of reverse engineering is to understand the concepts underlying an existing system, in order to exploit its functions to the full, facilitate knowledge transfer, improve the possibilities of maintenance and the effectiveness of reengineering and, finally, reuse the experience which has accumulated during the life of the system. Two problems arise in recovering the knowledge relative to an application: where to find the information and what level of abstraction is most suitable for representing the said knowledge.

The sources of information in an application system can be subdivided into two classes: static and dynamic.

The former includes the source code, the product documentation according to its standards (requirement specifications, design documentation, user manuals, test cases, etc.) and norms for use in user organizations. Static sources are often incomplete, ambiguous and disaligned.

Dynamic information sources, on the other hand, are constituted by the human resources interacting with the application: end users, programmers, designers, etc. These sources are characterized by their:
- great variability, as a subject memorizes the most frequently updated aspects but forgets those which have become obsolete, together with all their relative historical information;
- high diversification, as each subject only knows those aspects which directly concern him;
- low usability, as a large part of the useful information is informal in type and present in a free state in the human mind.

As regards the second problem, the knowledge representation aspect, in [5] four levels of software abstraction are described (conceptual, requirements, design and implementation), which can be used to describe both forward and reverse engineering processes.

The conceptual level describes a problem on general lines, in terms of a class of applications belonging to a certain domain. The requirements level provides greater details of a specific user problem belonging to the aforesaid class of applications. Both the conceptual and the requirements levels are independent of the solutions adopted. The design level describes the architecture of the solution in terms of components, relationships between components, data structures and algorithms. The solution is described independently of the technological platform available for setting it up. The implementation level, finally, contains all the details of the physical product as it is interpreted by the computer. A system must be described by different representations, each corresponding to a particular level of abstraction, but which must be traceable. The lower the level of abstraction, the greater the detailed information which must be handled to understand the working of the system. Generally, the user understands the representation at the conceptual level and, at most, at the requirements level.

In [1] is stated explicitly that reverse engineering cannot be completely automized because the essential information required for this task cannot be completely localized on static sources. Although this position is generally upheld, there is a positivist faith that the problem could be solved with the help of better technology, for example by means of static analyzers or expert systems minimizing the human intervention.

In [11] and [16], the use of static analyzers is proposed for extracting objects or entities from the implemented data structures. However, the entities produced with these analyzers are of a low level and lack any conceptual content, unless human intervention is invoked. Very often, in fact, modifications which do not conform to the initial project end up hiding the conceptual model, as these entities are spread through the physical data structures. The representation of the software system produced by these kind of analyzers is not very much more readable and meaningful than the code itself. For example, figure 1 represents the Entity-Relationship (ER) diagram constructed by a static analyzer relative to a file, whose Data Division is partially illustrated in figure 2. The diagram constitutes a different representation from the file one, but it does not add information, nor is the simplification more readable for the programmer or system user.

Although expert systems, like the one suggested in [1] and proposed in [2], also use dynamic sources, we do not consider them effective. The potential of expert systems derives not from inference mechanisms with a greater or lesser degree of sophistication but from a knowledge base which reflects the real world. In the case of reverse engineering, it is this very knowledge which is lacking, since the data structures of a real application are the result of the stratification of design decisions, whose effects only are known and not the reasons behind them.

The approach used by the authors in [10], which consisted of the construction of a logical data model, cleaned out from the already obtained design decisions, was also found not to be very incisive. The application of this method in an industrial environment showed that the data model thus obtained is no help to the user, because it is too far from the conceptual level, nor to the maintenance programmer, because the normalized logical model has gone away too greatly from the physical configuration.
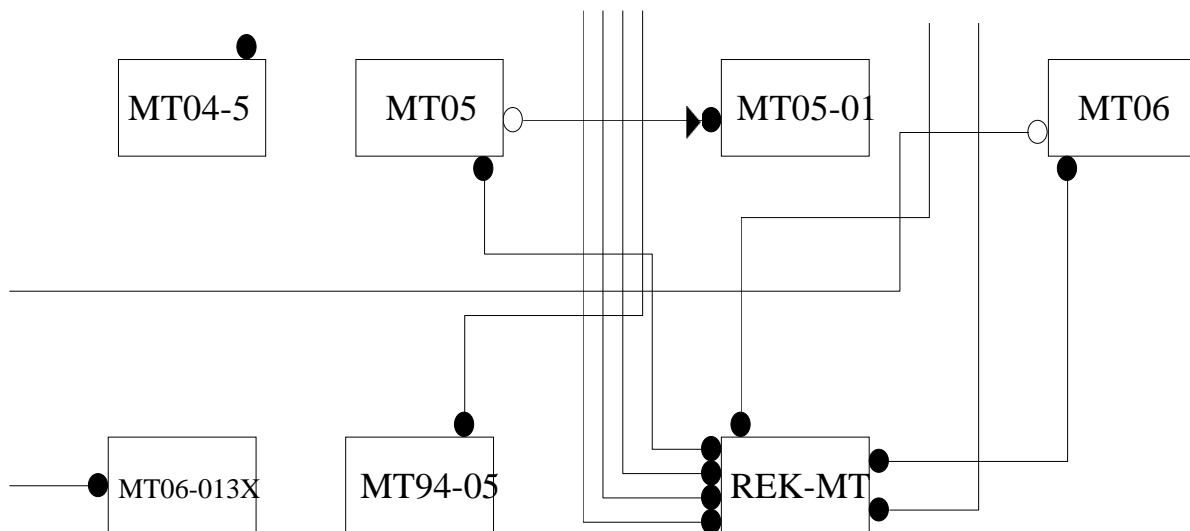

Figure 1. ER diagram of ARKMT file

```
000006*
000007****************************************************
000008* RECORD BEGINNING FILE
000009****************************************************
000010 01 REK-MT.
  .       .       .       .       .       .
000218          03   MT94-05.
000219*                             FIXED.
000220          05 MT94-051 PIC S9(5) USAGE COMP-3.
000221*                             PERCENT.
000222          05 MT94-052 PIC 99V99.
000223*                             MINIMUM.
000224          05 MT94-053 PIC S9(5) USAGE COMP-3.
000225*                             MAX.
000226          05 MT94-054 PIC S9(5) USAGE COMP-3.
000227*               DATE PASSING EXPENSE.
000228          03   MT04-06  PIC 9(6).
  .       .       .       .       .       .
000260          03   MT04-05.
000261*                             FIXED.
000262          05 MT04-051 PIC S9(5) USAGE COMP-3.
000263*                             PERCENT.
000264          05 MT04-052 PIC 99V99.
000265*                             MINIMUM.
000266          05 MT04-053 PIC S9(5) USAGE COMP-3.
000267*                             MAX.
000268          05 MT94-054 PIC S9(5) USAGE COMP-3.
000269*
000270          03   FILLER  PIC X(01).

000271               03   FILLER  PIC X(15).
000272*------- RECORD FIELDS PARTIAL PAYMENTS ---------
000273       02    MT05      REDEFINES MT00-R.
000274*   TABLE OF 05 PARTIAL PAYMENTS
000275          03   MT05-01        OCCURS 05 TIMES.
000276*                       DELAY PAYED
000277             05  MT-05-011 PIC S9(7)  COMP-3.
  .       .       .       .       .       .
000289*          03   FILLER  PIC X(04).
000290          03   FILLER  PIC X(34).
000291*----- RECORD FIELDS RATE VARIATION --------------
000292       02    MT06      REDEFINES MT00-R.
000293*   TABLE OF 06 RATE VARIATION
000294          03   MT06-01  OCCURS 06 TIMES.
000295*                 DATE RATE VARIATION
000296             05 MT06-011  PIC S9(6) COMP-3.
000297*                             DATE PASSING RATE
000298             05 MT06-012  PIC S9(6) COMP-3.
000299*                       NEW RATE
000300             05  MT06-013X.
000301              07 MT06-013  PIC 99V9999.
000302*                             INT.CRED./TRANSF.
000303             05 MT06-014  PIC S9(7) COMP-3.
000304*                     INT UPD./TOT TRANSF.
000305             05 MT06-015  PIC S9(7) COMP-3.
000306*
000307*          03   FILLER  PIC XX.
000308          03   FILLER  PIC X(32).
000309****************************************************
```

Figure 2. Data Division of ARKMT file

In this study we contend that data reverse engineering at the conceptual level is an activity in which human intelligence plays the preponderant role, while the role of technology, albeit useful, is only marginal. While even the effort to reach a design abstraction level which is really independent of the realization is considerable, as discussed in [9] and [13], the chasm between the conceptual abstraction level and the implementation abstraction level is so great that the heaviest effort is dedicated to tracing back the conceptual formulation of the problem.

To tackle the essence of the problem of recovering data at the conceptual level, it is necessary to change the method used up to now. Instead of a purely bottom-up approach, starting exclusively from the source code, in this study we propose a reverse engineering method which integrates analysis of the static sources, automatically performed in bottom-up fashion, with consultation of dynamic sources, performed manually in top-down fashion, as it must be guided by precise questions which are the fruit of a series of successive refinements. The starting point for this method is the use of a reference information model which describes conceptually the class of the application domain.

This paper continues as follows: section 2 describes the data reverse engineering method we propose, section 3 describes the experimentation of the method in a banking application, section 4 discusses the lessons learned from the experimentation and, finally, section 5 draws some conclusions and outlines future development.

2. Method

Reverse engineering can be applied to obtain specifications of data, processes or both. As we have already pointed out in [Como91], data reverse engineering and process reverse engineering must proceed by repeated bootstraps, as a lot of information extracted from data is useful in process analysis and vice versa. We feel that in data-oriented applications like business systems, however, the data contain a large part of the knowledge which is useful both for data reverse engineering and process reverse engineering. In fact, a typical information system contains thousands of data elements but also millions of code lines, of which little is known. Recovering the knowledge of these data increases the learning speed of the many programmes using them. Furthermore, data problems have repercussions on all the programmes which access external data. Figure 3 models our method for the process of data reverse engineering, whose steps are described in the following.
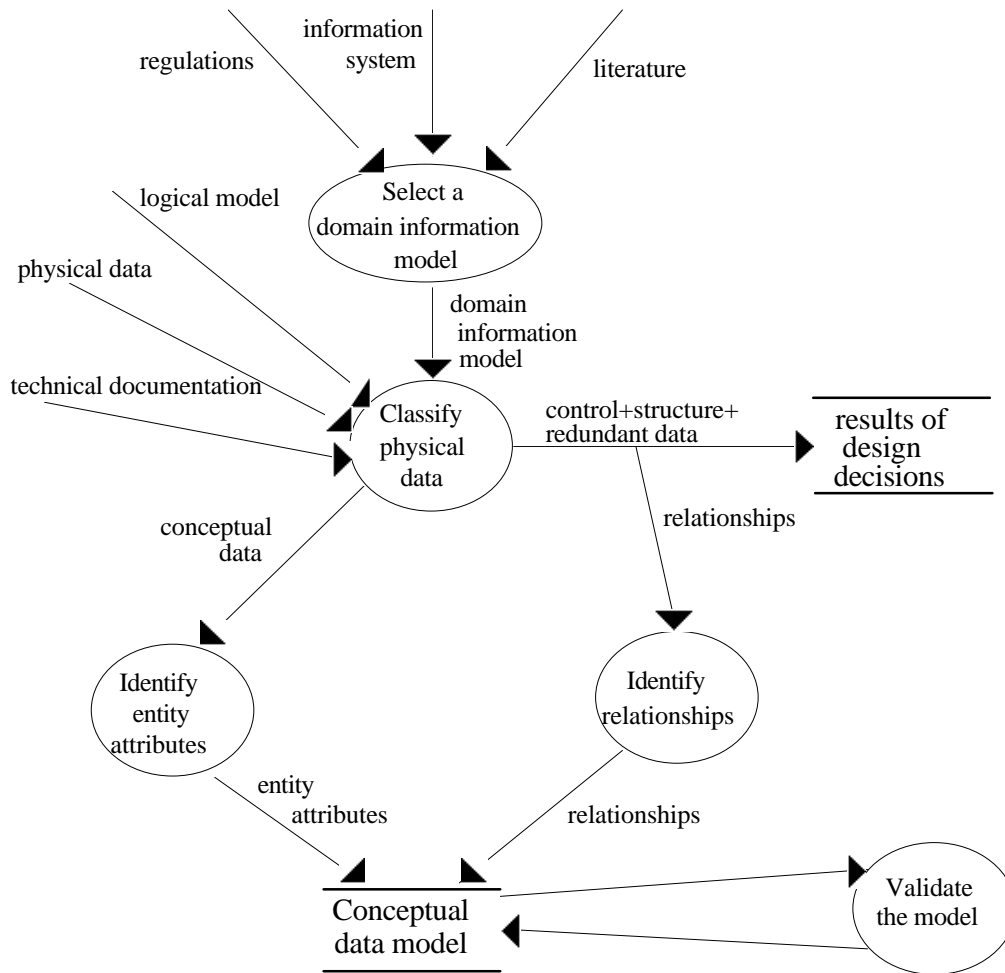
Figure 3. Process model of data reverse engineering

2.1. Selection of the domain information model

This step involves the individuation of a *reference information model* with a high abstraction level, which can be applied to the application domain. The information model in question must be both simple, for an easy comprehension, and general, so that it can be applied to all the information systems belonging to that class of problems. Information sources for defining the domain information model derive from:

- literature on the application domain, for instance books on banking techniques;
- norms and laws in force, such as the internal organizational norms of users and provisions laid down by controlling government bodies;
- existing software systems in the application domain.

The information model thus obtained is not only useful in the reverse engineering process but provides the user with a reference system on the evolution of the information system. In fact, some parts of the domain information model might not have counterparts in the current information system and this would constitute a possible area for extending the system.

The information model used as reference model contains entities and entity hierarchies, which define the application domain for a whole class of problems, but not all the association relationships. In fact, while some relationships, such as CLIENT. initiates .TRANSACTION and INSTALMENT. pays off part of. LOAN are typical of the application domain, many others express the data usage strategy and therefore depend on the user, not only on the applicative domain. The relationships the user sees between entities in the domain information model can be inferred from the design decisions, whose effects are, however, scattered in the software system, both in the data and the programmes. Thus a part of the relationships can be recovered with data reverse engineering while for the rest, process reverse engineering must be performed.
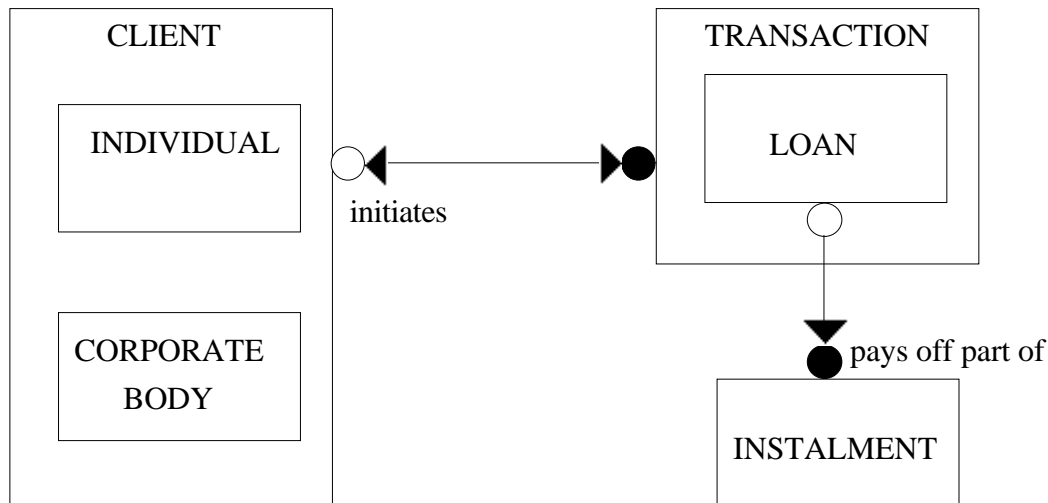
Figure 4. The reference information model

The definition of the reference model is of the incremental type. It provides information on the data expected from the reverse engineering but it may also be the reverse engineering process of the current system to provide information integrating the reference model. Figure 4 shows the ERD used as information model relative to the bank loans domain. An entity CLIENT with two subentities INDIVIDUAL and CORPORATE BODY can be observed, as well as an entity TRANSACTION with a subentity LOAN, and an entity INSTALMENT.

The reference information model can also be extended to the requirements abstraction level. In this case, the attributes expected by the entities are also described.

2.2. Classification of physical data

By analyzing the content of the data files or the database, in data-oriented applications it is possible to distinguish the following information.
- *Conceptual data*: the data belonging to the application domain. These data are often codified with cryptic names and appear several times under different names or with different formats. The meaning of an item of conceptual data is only discovered after iterative consultation of static sources and experts in the application domain. If necessary, a meaningful name must be assigned and the result of the recognition must be placed in the data dictionary.
- *Control data*: the data which record a past event and are used to control the logic of programmes accessing them. For example, a flag to indicate the logical cancellation of a record.
- *Structure data*: the data which organize data in data structures according to the potential offered by the

programming environment. For example, record pointers.

Control and structure data are the result of design decisions taken at different times by different subjects with variable limitations. Often, in static sources, only the result of these decisions is present, without the rationale which determined them. The role of these data must be discovered by consulting the designers and programmers and it must then be documented in the data dictionary.

2.3. Identification of entity attributes

Conceptual data are attributed to the entities in the reference information model. In the case of hierarchies of entities, the data are placed at the highest level possible, as subentities inherit the attributes of the superentity. The attributes, too, are defined stepwise. When some recovered data are not included among the expected data, they are added to the reference model at the requirements abstraction level. If, on the other hand, expected data are not found in the existing system, they provide indications as to how the system could be extended. Data which can be calculated from other primitive conceptual data are recognized as derived data, and in the data dictionary the derivation algorithm is included in the definition. Derived data are not essential for the definition of the conceptual data model but can be useful because they generate expectations on the calculation functions in the processes.

2.4. Identification of relationships

Control, structure and redundant data can help to identify the association relationships of static type between the entities in the preliminary information

model. For example, the structure data present in the record layouts of the application can create a navigational path used for implementing a relationship.

2.5 Validation of the conceptual data model

The conceptual data model, including all the definitions in the data dictionary, must be revised with the user. This step is identical with the one performed by the analyst with the user in a traditional software development paradigm.

3. Experimentation

The experimentation carried out belongs to a wider project for recovering information assets in an Italian bank. The goal is to use a consistent and readable documentation both for ordinary maintenance activities and as a base for successive renovation tasks.

The information system works on a Siemens mainframe with BS2000 operating system and is constituted by Cobol programmes operating on an indexed sequential file system. In particular, for this experiment the loan management subsystem was chosen, which uses 4 data files with 190 record types and 700 data items.

Before proceeding with our proposed method, a static analyzer was applied, which automatically produces a logical model in the form of an ER diagram, already illustrated in figure 1, whose attributes are stored in the data dictionary.

Table 1 shows the statistics relative to the classification of physical data step. The 287 redundant data in the table do not produce further entries in the new data dictionary.

Table 2 shows the results produced by the identification of the entity attributes step. Only the specific attributes of an entity are counted in the table and not those inherited from hierarchically superior entities.

The identification of the relationships step did not lead to the definition of new relationships but made it possible to verify the cardinal and mandatory nature of the two expected relationships: CLIENT. initiates .TRANSACTION and INSTALMENT. pays off part of. LOAN. In figures 5 and 6 it is possible to compare the state of the old record layouts with the new data dictionary with its corresponding definitions. For the sake of clarity, the descriptions have been translated from the original version. The data dictionary contains the information for localizing the physical data in the data files, so to ensure the traceability with implementation.

As regards the technology, the experiment was supported by X-ref files-programmes tools, a data extractor which populates the data dictionary starting from the source code, and a data dictionary manager for browse and edit tasks.

The experiment was performed by two reverse engineers over a period of three months, for a total of 370 man-hours, 85 of which were devoted to technical support activities. An expert in the banking domain was occasionally consulted, for a total of 8 hours. The reverse engineers' productivity was initially 2 recovered data items/hour and later 3 data items/hour. The improvement in productivity was due to the reuse of previously recovered knowledge.

Conceptual data	Control data	Structure data	Redundant data
197	36	180	287

Table 1. Results from classification of physical data

ENTITY	Attributes Found
CLIENTE	30
INDIVIDUAL	22
CORPORATE BODY	23
TRANSACTION	16
LOAN	79
INSTALMENT	27

Table2. Results from identification of entity attributes

Field Name	Type	LL	M/O M=Mandatory O=Optional	Description
....	....	.....	.....	.....
....	....	....	....	....
ARKAG file - record type 1				
Field Name	**Type**	**LL**	**M/O** **M=Mandatory** **O=Optional**	**Description**
....	.....	.....	.....	.....
....	.....	.....	.....	.....
AG22E	N	11	O	Risks centre code
....	.....	.....	.....	.....
AGN-L1	N	5	O	Code-ABI
....	.....	.....	.....	.....
....	.....	.....	.....	.....
ARKMT file - record type 000				
Field Name	**Type**	**LL**	**M/O** **M=Mandatory** **O=Optional**	**Description**
....	.....	.....	.....	.....
....	.....	.....	.....	.....
MT03-21	N	1	M	instalment paid flag 0=instalment not paid 1= instalment partially paid
....	.....	.....	.....	.....
MT00-2	N	6	M	Last update
....	.....	.....	.....	.....
....	.....	.....	.....	.....
TAB025 file - record type 00				
Field Name	**Type**	**LL**	**M/O** **M=Mandatory** **O=Optional**	**Description**
....	.....	.....	.....	.....
....	.....	.....	.....	.....
TB65	N	2	M	Area index enpowered branches
....	.....	.....	.....	.....
TB68	A	3	M	Segment number TP0000
....	.....	.....	.....	.....
....	.....	.....	.....	.....

Figure 5. Old technical documentation

Data item name	Description	Classification	Dimension	Domain	Physical name
....	....	....	.....	....	....
CODE-ABI	Italian-Banking-Association code used when client is a a credit company	Conceptual entity CORPORATE BODY	code	numeric-code5	AGN-L1 file ARKAG record type 1
....	....	....	.....	....	....
CODE-RISK	Code assigned by risks centre to client having one or more credit limits from any bank on national territory	Conceptual entity CLIENT	code	code11	AG2E file ARKAG record type 1
....	....	....	.....	....	....
....	....	....	.....	....	....
DATE-UPD-ADD-INSTALMENT	Date for comparison with instalment due date for debiting relative sum to corrent account. Date is introduced by operator on activation of a batch procedure. This procedure operates on loans with automatic debiting order.	Control			MT00-2 file ARKMT record type 000
....	....	....	.....	....	....
DATE-PAYMENT-INSTALMENT	Indicator of unpaid or partially paid instalment	Control			MT03-21 file ARKMT record type 000
....	....	....	.....	....	....
....	....	....	.....	....	....
POINTER-MOV-C/A-S/D	Pointer signalling entry for transactions programme for current accounts and savings deposits	Structure			TB68 file TAB025 record type 00
....	....	....	.....	....	....
POINTER-BRANCH-AREA	Index of branches area	Structure			TB65 file TAB025 record type 00
....	....	....	.....	....	....
....	....	....	.....	....	....
....	....	....	.....	....	....
....	....	....	.....	....	....

Figure 6. New data dictionary

4. Lessons learned

It should be noted that the experience presented was gained on a real, large-sized software system, analyzing one of its subsystems. Hence we feel that the lessons learned can be extended to other large systems. The lessons learned are summarized below:

- *Static sources do not suffice for the reverse engineering process.*

Technical manuals were found lacking (references to many data present in the files were missing) and also unreliable (some data description were different from the real significance). The usage manuals were found to be up-to-date, but only as regarded currently used parts, with no indication of the obsolete data. Legal regulations are often not directly evident in the database setup, because they have been complied with by expedients included in the procedures and we will expect to find them in the process reverse engineering.

- *Dynamic sources are individual, incomplete and geographically scattered.*

The system's great diffusion results in many users and maintenance points but the overall inclusion of dynamic sources provides up-to-date information on the system's status and a good level of awareness of its history. The fragmentary nature of the information makes the construction of updated documentation necessary for the survival of the system. The intensive use of dynamic sources and the importance in recovering software assets confirmed the marginal role of automatic instruments in data reverse engineering.

- *The conceptual content is relatively low in a software system with respect to the data for making the program work.*

As shown in table 1, the disproportion is due to the overhead caused by design decisions taken during maintenance activities. Many could probably be eliminated by accurate reengineering but, in any case, software maintenance should become more disciplined to lengthen the system's life.

- *Knowledge accumulated in an application system is transferable.*

The two reverse engineers involved in the experiment did not know the application system, nor were they experts in the banking domain. As a result of the application of our method, they mastered both a general knowledge of bank loans and a specific knowledge of the existing application. This shows that it is possible to concentrate and transfer knowledge.

5. Conclusions and future research

The data reverse engineering method proposed in this study is essentially based on the disciplined use of human resources to enrich and populate a reference information model. Experimental comparison of our method with other approaches which emphasize the use of reverse engineering technologies revealed that the information necessary for maintaining the application is not obtained automatically but resides in the human intelligence. To guide the latter, models expressed at conceptual level, like the reference information model used in our method, are necessary.

The information obtained from data reverse engineering is what is effectively required for maintaining the application. By reading the documentation produced, the user can recover his fund of information, while in the case of modification, the programmer can evaluate whether he is concerned with the requirements, the structure or the control aspect.

Data restructuring in order to normalize the data structures was not performed as, unlike reverse engineering, reengineering of data must be carried out at the same time as the reengineering of procedures. In fact, it is possible to maintain the traceability of the data models at different levels of abstraction while movement is exclusively in the reverse (abstraction) or forward (refinement) directions, but when system changes (alteration) are made, the traceability must be defined with the new target system, which is composed of both data and procedures.

Future research will aim to extend the scope of the experimentation by analyzing the other data files of the banking information system and studying the econometrics of the process, as a means of assessing when the application of effort is rewarded by added value emerging from the information obtained.

Acknowledgements

References

[1]   C.Bachman, "A CASE for reverse engineering", Datamation, July 1, 1988, pp.49-56.

[2]   P.Benedusi, V.Benvenuto, and M.G.Caporaso, "Maintenance and prototyping at the entity-relationship level: a knowledge-based support", *Proceedings of the COnference on Software Maintenance 1990*, San Diego, California, IEEE Computer Society Press, 1990, pp.161-169.

[3]   T.J.Biggerstaff, "Design recovery for maintenance and reuse", *IEEE Computer*, July 1989, pp.36-49.

[4]   P.T.Breuer, and K.Lano, "Creating specifications from code: reverse engineering techniques", *Software Maintenance: Research and Practice*, vol.3, 1991, pp.145-162.

[5]   E.J.Byrne, "A conceptual foundation for software re-engineering", *Proceedings of the Conference on Software Maintenance 1992*, Orlando, Florida, IEEE Computer Society Press, 1992, April 1987, pp.226-235.

[6]   G.Canfora, and A.Cimitile, "Reverse engineering and intermodular data flow: a theoretical approach", *Software Maintenance: Research and Practice*, vol.4, 1992, pp.37-59.

[7]   E.J.Chikofsky, and J.H.Cross, "Reverse engineering and design recovery: a taxonomy", *IEEE Software*, January 1990, pp.13-17.

[8]   A.Cimitile, Towards reuse reengineering of old software", *Proceedings of the Fourth International Conference on Software Engineering and Knowledge Engineering*, IEEE Computer Society Press, Capri, Italy, 1992, pp.140-149.

[9]   A.Cimitile, F.Lanubile, and G.Visaggio, "Traceability based on design decisions", *Proceedings of the Conference on Software Maintenance 1992*, Orlando, Florida, IEEE Computer Society Press, 1992, pp.309-317.

[10] G.Como, F.Lanubile, and G.Visaggio, "Design recovery of a data-strong application", *Proceedings of the Third International Conference on Software Engineering and Knowledge Engineering*, Skokie, Illinois, 1991, pp.205-212.

[11] H.P.Haughton, and K.Lano, "Objects revisited", *Proceedings of the Conference on Software Maintenance 1991*, Sorrento, Italy, IEEE Computer Society Press, 1991, pp.152- 161.

[12] *IEEE Software*, special issue on "Maintenance and reverse engineering", January 1990.

[13] F.Lanubile, and G.Visaggio, "Maintainability via structure models and software metrics", *Proceedings of the Fourth International Conference on Software Engineering and Knowledge Engineering*, IEEE Computer Society Press, Capri, Italy, 1992, pp.590-599.

[14] J.A.Ricketts, J.C.DelMonaco, and M.W.Weeks, "Data reengineering for application systems", *Proceedings of the Conference on Software Maintenance 1989*, Miami, Florida, IEEE Computer Society Press, 1989, pp.174-179.

[15] H.M.Sneed, and G.Jandrasic, "Inverse transformation of software from code to specification", *Proceedings of the Conference on Software Maintenance 1988*, Phoenix, Arizona, IEEE Computer Society Press, 1988, pp.102-109.

[16] H.M.Snedd, "Migration of procedurally oriented Cobol programs in an object-oriented architecture", *Proceedings of the Conference on Software Maintenance 1992*, Orlando, Florida, IEEE Computer Society Press, 1992, pp.105-116.

[17] M.Ward, F.W.Callis, and M.Munro, "The maintainer's assistant", *Proceedings of the Conference on Software Maintenance 1989*, Miami, Florida, IEEE Computer Society Press, 1989, pp.307-315.