

# Analyzing the Application of a Reverse Engineering Process to a Real Situation

Fabio Abbattista (\*), Gregorio M.G. Fatone (\*\*), Filippo Lanubile (\*), Giuseppe Visaggio (\*)

(\*) Dipartimento di Informatica, University of Bari, Italy

(\*\*) Basica S.p.A., Potenza, Italy

## Abstract

*A reverse engineering process model was applied and, on the basis of the data collected, some modifications were made aiming to improve its efficacy.*

*The experience gave rise to various considerations of interest, first among them being the clear interaction between the quality of the product and the quality of the process. A method of synergetic application of static and dynamic analysis to improve understanding of the program was consolidated. The experience enabled modifications to be introduced connecting the reverse engineering process more closely with the understanding of the programs and information deriving from the application domain.*

*Finally, the problem of the efficacy of the tools used to obtain the reverse engineering products was made evident during the experimentation on the field.*

## 1: Introduction

We present an experience in which process quality and product quality interact and mutually improve one another. The process is reverse engineering while the product is the documentation of programs necessary to exploit the program better. The salient point to be gained from the experience are, in general, the model as it appears after improvements stemming from trial on the field and, in particular, the method for integrating static and dynamic analyses to improve the process.

The paper describes the application of a process model to a real situation. The scenario is a set of programs with the following characteristics:

language:	COBOL
operating system:	BS2000
total no. of programs:	653
no. of on-line programs:	348
no. of batch programs:	305
no. of files:	70
no. of data:	9 000
no. of Instructions:	900 000

(including comments but not copy books)

These programs constitute a "large" software system as they are integrated by a database and cover various banking business areas. Their long life (average 12 years, reaching peaks of 23 years) was mirrored by their "old" structure, so that they were difficult to maintain and had inconsistent, invariably poor documentation. The system is so diffuse within the user bank that substituting it would have been impracticable. It was therefore necessary to rejuvenate the software system and, as the only reliable component was the source code, it was judged necessary to start by reverse engineering to understand the programs, while acknowledging that reengineering would then be required.

The paper includes a brief description of the process model (section 2), the results of its application (section 3), the modifications made to the process model (section 4) and the final conclusions (section 5).

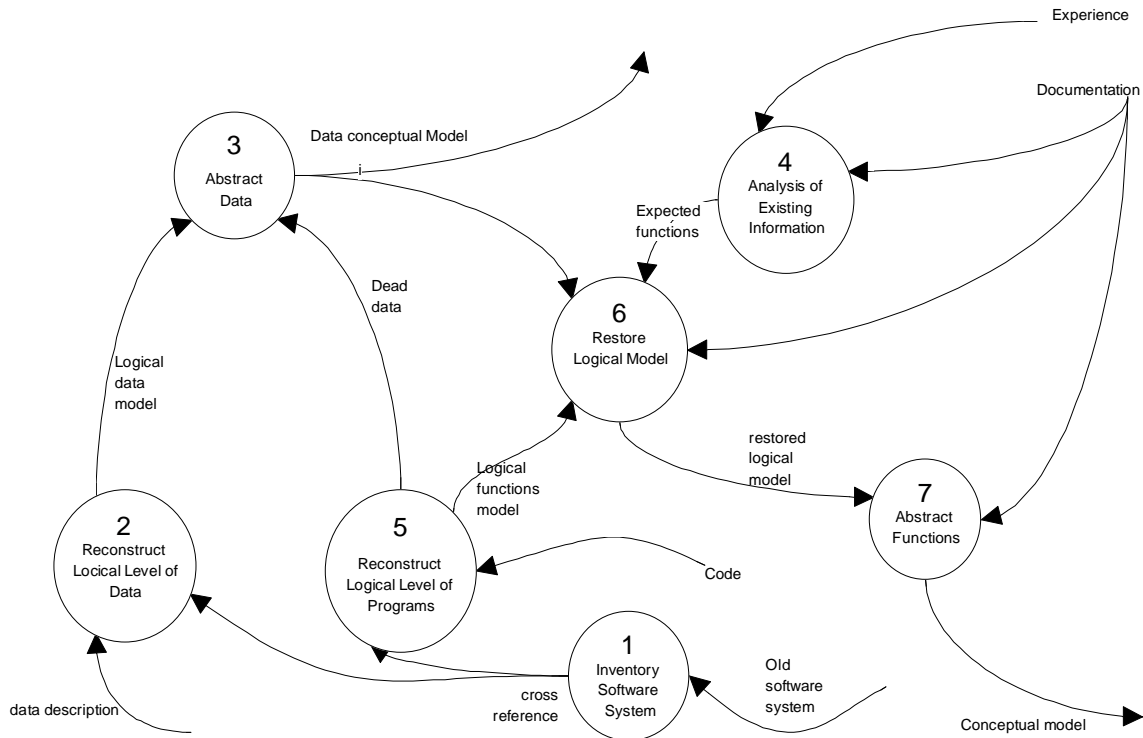
## 2: The reverse engineering process

Our reverse engineering process had two main objectives: (1) to increase the ease of maintenance of the software system, and (2) to improve its usability by the final users and the ease of knowledge transfer among different users [5].

The first involved reconstruction of the project design documentation and restoration of the most degraded parts while the second required reconstruction of the user documentation and the data conceptual model. This helps users to understand their own information system better from the point of view of the data processed [4], [6].

Figure 1 shows the process model, which is briefly described underneath. Further details may be obtained from [10].

1. Inventory Software System. The following cross references are extracted from the old software system: call dependence X-ref, copybook X-ref, and file access X-ref.



**Figure 1. The process model**

**2. Reconstruct Logical Level of Data.** From the data description, the hierarchical structure constituting the logical data model is reconstructed. The aliases are recognizable entities in the application domain. Figure 2 shows a record declaration, as input to the phase, and the relative hierarchical diagram, as output from the phase. The arrows represent the hierarchical relationships between the substructures in the record; elementary data are not shown. The data are classified as application domain data, control data, and structural data. Application domain data are the attributes of recognizable entities in the application domain. For example, the field named MT02-02 representing the "amount" is also an attribute of MORTGAGE and is therefore recognizable in the application domain. Control data have no correspondence with the application domain but are used to record the occurrence of an event during the execution of a program, so that other programs can adapt accordingly; flags validated by one program and used by others, asynchronously, to determine their behavior according to the previous history of the software system, are typical examples of

control data. For example, the field MT02-33 is preset to indicate the existence of an agreement for taking out the mortgage, which must form the basis of the variation in interest rate to be applied when calculating the instalment; it is a typical example of control data. Structural data are data necessary for managing the organization of data bases or files. The field MT01-05 is a typical example of structural data, because it identifies the record type inside the MORTGAGE file.

**3. Abstract Data.** All the data in the application domain which belong to the logical model and are not dead are associated with the corresponding meaningful concept for the application domain [1].

**4. Analysis of Existing Information.** This activity involves identifying the expected functions in the program being reversed using two types of information. The first is static knowledge, i. e. the internal and external documentation of the rules governing the application domain of the function. The second is dynamic, derived from the experience of the programmers and users who interact with the working programs.

```

01REK-MT.
02 N PIC X.
02 MPIC 9(6).
02 MREDEFINES MT00
03 MOT01-1.
04 MT01-01PIC 99.
04 MT01-02 PIC 99.
04 MT01-03 PIC 99.
04 MT01-04 PIC 9(6).
03 MT01-05 PIC 999.
02 MT00-R.
03 MT00-1PIC 9(6).
03 MT00-2 PIC 9(6).
03 MT00-3 PIC 9(6).
03 FILLER PIC X(146).
02 MT02 REDEFINES MT00-R.
03 MT02-01 PIC X(16).
03 MT02-02 PIC 99(13) USAGE COMP-3.
03 MT92-03 PIC 99V99.
03 MT02-04 PIC 999.
03 MT02-05 PIC 999.
04 MT02-061PIC 999.
04 MT02-062 PIC 9(6).
03 MT02-07 PIC 9(6).
03 MT02-08 PIC S9(13) USAGE COMP-3.
03 MT02-09 PIC S9(13) USAGE COMP-3.
03 MT02-10 PIC S9(9) USAGE COMP-3.
03 MT02-11 PIC 9.
03 MT02-12 PIC 9.
03 MT02-13 PIC 99.
03 MT02-14 PIC X(4).
03 MT02-15 PIC 9.
03 MT02-16 PIC 9.
03 MT02-17 PIC 9.
03 MT02-18 PIC X.
03 MT02-19 PIC 99.
03 MT02-22 PIC 9.
03 MT02-23 PIC 9.
03 MT02-30 PIC 9.
03 MT02-24 PIC 9.
03 MT02-25 PIC 99.
03 MT02-26 PIC 9.
03 MT02-27 PIC S9(7) COMP-3.
03 MT02-28 PIC S9(7) COMP-3.
03 MT02-29 PIC S9(7) COMP-3.
03 MT02-03 PIC 99V9999.
03 MT02-31PIC 99V99.
03 MT02-32.
05 MT02-321PIC 99V9(4).
05 MT02-322 PIC 99V9(4).
03 MT02-33 PIC 9.
03 MT02-34 PIC X(4).
03 MT02-35 PIC XX.
03 FILLER PIC X(18).
03 MT02-36BAS PIC 9.
03 MT02-37BAS PIC 9(16).

02 MT03 REDEFINES MT00-R.
03 MT03-01PIC 9(6).
03 FILLER REDEFINES MT03-01.
05 MT03-01AA PIC 99.
05 MT03-01MM PIC 99.
05 MT03-01GG PIC 99.
03 MT03-02 PIC S9(11) USAGE COMP-3.
03 MT03-03 PIC S9(11) USAGE COMP-3.
03 MT03-04 PIC S9(11) USAGE COMP-3.
03 MT03-05 PIC S9(11) USAGE COMP-3.
03 MT03-06 PIC S9(11) USAGE COMP-3.
03 MT03-07 PIC S9(5) USAGE COMP-3.
03 MT03-08 PIC S9(11) USAGE COMP-3.
03 MT03-09 PIC 9(6).
03 MT03-10 PIC S9(9) USAGE COMP-3.
03 MT93-11PIC S9(6) USAGE COMP-3.
03 MT03-12 PIC 9.
03 MT93-13 PIC 99V99.
03 MT93-14 PIC 99V99.
03 MT03-15 PIC S9(7) USAGE COMP-3.
03 MT03-16 PIC 99.
03 MT03-17 PIC 9(6) COMP-3.
03 MT03-18 PIC S9(7) COMP-3.
03 FILLER PIC X(02).
03 MT03-19 PIC S9(7) COMP-3.
03 MT03-20 PIC S9(7) COMP-3.
03 MT03-21PIC 9.
03 MT03-22 PIC S9(7) COMP-3.
03 MT03-23 PIC 9(6).
03 MT03-13 PIC 99V9999.
03 MT03-14 PIC 99V9999.
03 MT03-24 PIC X.
03 MT03-11PIC S9(7) USAGE COMP-3.
03 FILLER PIC X(40).
03 MT03-25 PIC 9(4).
02 MT04 REDEFINES MT00-R.
03 MT04-01.
05 MT04-011PIC 99.
05 MT04-012 PIC 99.
05 MT04-013 PIC 99.
05 MT04-014 PIC 9(6).
05 MT04-016 PIC 99.
03 MT04-02.
05 MT04-021PIC 99.
05 MT04-022 PIC 99.
05 MT04-023 PIC 99.
05 MT04-024 PIC 9(6).
03 MT94-03 PIC 99V99.
03 MT04 PIC 9(6).
03 MT94-05.

05 MT94-051PIC S9(5) USAGE COMP-3.
05 MT94-052 PIC 99V99.
05 MT94-053 PIC S9(5) USAGE COMP-3.
05.MT94-054 PIC S9(5) USAGE COMP-3.
03 MT04-06 PIC 9(6).
03 MT94-07 PIC 99V99.
03 MT04-08 PIC S9(11) USAGE COMP-3.
03 MT04-09 PIC 9(6).
03 MT04-10 PIC 99.
03 MT04-12 PIC 9(9) USAGE COMP-3.
03 MT94-11PIC S99V99.
03 MT04-13 PIC S9(9) COMP-3.
03 MT04-14 PIC 9(6).
03 MT04-15 PIC S9V9999.
03 MT04-16 PIC 9(6).
03 MT04-17 PIC S9(9) COMP-3.
03 MT04-18 PIC 9(3).
03 MT04-03 PIC S9V9999.
03 MT04-07 PIC S9V9999.
03 MT04-11PIC S99V9999.
03 MT04-05.
05 MT04-051PIC S9(6) USAGE COMP-3.
05 MT04-052 PIC S9V99.
05 MT04-053 PIC S9(7) USAGE COMP-3.
05 MT04-054 PIC S9(7) USAGE COMP-3.
03 FILLER PIC X(01).
03 FILLER PIC X(11).
03 MT04-056 PIC 9(4).
02 MT05 REDEFINES MT00-R.
03 MT05-01OCCURS 05 TIMES.
05 MT05-011PIC S9(7) COMP-3.
05 MT05-012 PIC S99999 COMP-3.
05 MT05-013 PIC S9(9) COMP-3.
05 MT05-014 PIC S9(9) COMP-3.
05 MT05-015 PIC S9(9) COMP-3.
05 MT05-016 PIC S9(6) COMP-3.
03 FILLER PIC X(34).
02 MT06 REDEFINES MT00-R.
03 MT06-01OCCURS 06 TIMES.
06 MT06-011PIC S9(6) COMP-3.
06 MT06-012 PIC S9(6) COMP-3.
06 MT06-013X.
07 MT06-013 PIC S9V9999.
06 MT06-014 PIC S9(7) COMP-3.
06 MT06-016 PIC S9(7) COMP-3.
03 FILLER PIC X(32).

```

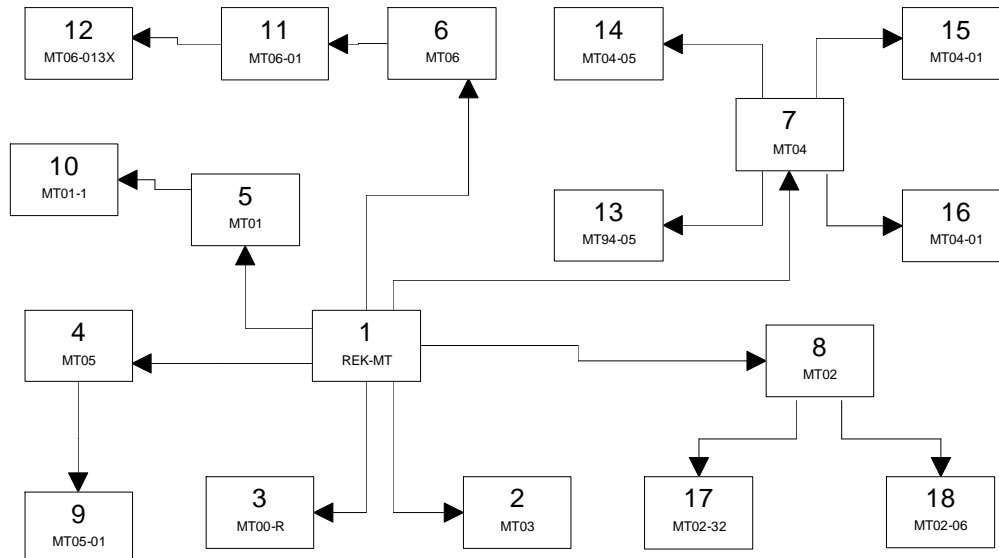
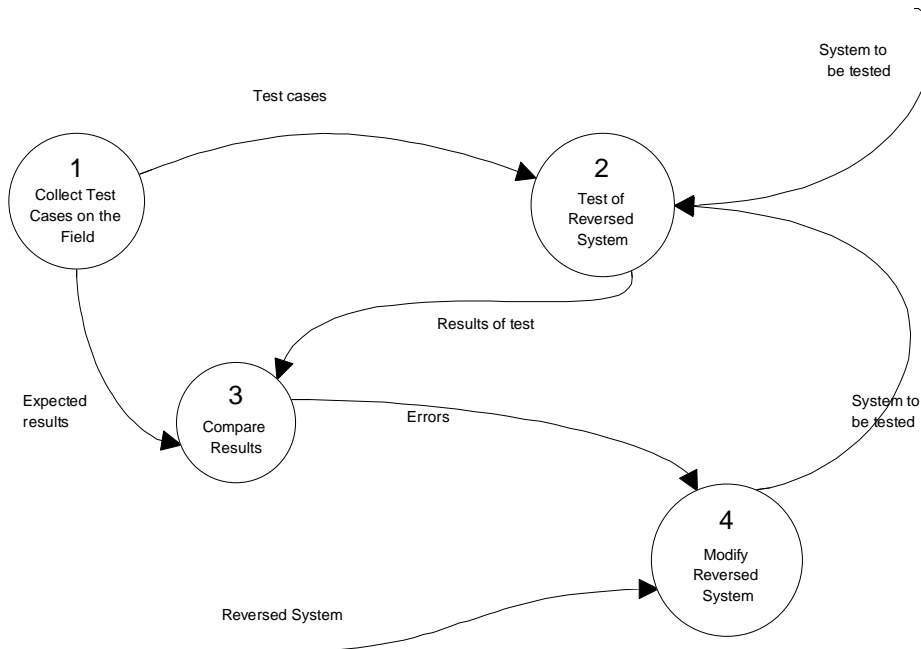


Figure 2. Input and output of "Reconstruct Logical Level of Data" phase



**Figure 3. Equivalence test of the system before and after reverse engineering**

5. Reconstruct Logical Level of Programs. Each program is associated with a structure chart in which each module corresponds to a SECTION or an external subroutine of the program. In this phase, both dead data and dead instructions are identified. These are data not used by the program and instructions which cannot be run, respectively. The former are communicated to the "Abstract Data" activity while the latter are erased from the structure chart, which thus constitutes the logical model of the functions.

6. Restore Logical Model. Restoration involves introducing changes to improve the structure of the programs and make them easier to maintain, without causing repercussions on the data or interfaces with other systems. Some examples of modifications are renaming of variables, making their identifiers more meaningful; extracting modules with high internal cohesion from those with low cohesion and isolating them in the structure ([7], [8], [12], [13]); externalizing modules which, in the present process, are in line with the main; localizing variables declared to be global but used locally in both existing modules and in processes extracted during restoration. Execution of these activities is facilitated by the expected functions derived from the phase of analysis of existing information. In fact, thanks to this knowledge, the operators can extract the functions

from the modules present in the logical model. This makes the logical model more readable and its modules less complex.

7. Abstract Functions. The functions abstracted during restoration are documented. The aim of each function is described in textual form. The relationships between functions are also documented by means of data flow diagrams. The latter, together with the description of each function, constitute the conceptual model [9].

The reverse engineering process is not symmetrical because the programs are restored while the data are not, because any interference with the latter would affect the procedures and make the whole restoration process very expensive. In fact, restoration of the programs is confined to the instructions of each single program, a much simpler and more economical process.

The reverse engineering process described modifies code, so that it is necessary to verify that the working programs are equivalent to those produced by the process. A test process is used and, as the only reliable component in the working system is the code in question, it is only possible to test the equivalence between the actual and the reversed program. Test cases obtained during normal working of the actual system are used. The equivalence test is modeled in Figure 3.

R.E.of Processes	Total Effort (h.)	Total Locs.	Locs/h.	Total Procedure	Effort for	Effort for	Effort for Normalized	Other Effort (h.)
Resource				Division locs	Physical Level (h.)	Logical Level (h.)	Logical Level (h.)	
1° Reverse-Operator	386	55504	143	4439	89.5	37.5	196	63
2° Reverse-Operator	406	65201	160	10728	51	42	310.75	2.5
3° Reverse-Operator	377	49588	131	5292	86	18	186	87
4° Reverse-Operator	372	64304	172	6496	66	10	221	75
5° Reverse-Operator	306	27089	88	3157	80.5	9.5	147	69

**Table I. Measurements in process reverse engineering**

R.E. of Data Resource	Total Effort	Redocumented	Productivity	Effort for	Effort for	Other effort (h.)
		Data		Physical Level (h.)	Conceptual Level (h.)	
1° Reverse-Operator	1966	3838	1.95	145	1523	298
2° Reverse-Operator	1278	1671	1.31	51	1146	81
3° Reverse-Operator	1744	2904	1.66	101	1513	130

**Table II. Measurements in data reverse engineering**

### 3: Operative results

The planned process was put in production in the scenario described earlier and after seventeen calendar months of work on a production line, the first results were obtained.

It should be noted that a production line refers to an organizational unit which has all the resources required for executing the process autonomously. In this case, the production line is composed of eight reverse operators and one reverse engineer. The former execute the procedures according to the defined process models while the latter coordinates activities and takes all the decisions necessary for solving all indeterminate points in the execution procedures. The production line shares an expert in the application domain with other organizational units in the company.

These first operative results can be analyzed from the point of view of both efficiency and efficacy. Although efficiency was not the main aim of this work, the data on the activities for reverse engineering of the programs and of the data are summarized in Tables I and II, respectively. Two important considerations can be made.

The productivity of the operators for reverse engineering the programs is correlated with their experience in the application domain; this explains the differences seen in Table I. In reverse engineering the data, the difference in productivity has less correlation with experience because there is very little automation of

the activities and so the man time required is very high in any case.

The second point is that commercially available tools are often inadequate for large projects. For example, the tool used in extracting the data structure becomes unacceptably slow when access to previously inserted information is required, if the data are more than a thousand or if access is to an entity with more than one hundred attributes. The tool used for the programs, on the other hand, shows an abrupt drop in performance (answers to questions on data and control flow slow down) as soon as the threshold of 8000 lines of code is passed. Clearly, inefficient automatic tools require more man time to attain the objectives.

As regards the efficacy of the process, the following points can be made.

The logical data model is not very useful to the maintainer because he can read the information he needs more easily from the record layout described in code. This is due to the complicated structure of the files in the old system, which is difficult to clarify and represent. It is necessary, however, to know the relationships existing between the files that manage the system; there are many of these, so maintenance is a high risk procedure. For example, the relationships between the CUSTOMERS' INFORMATION file and the MORTGAGE: CLIENT TAKES OUT MORTGAGE is expressed in the field MT02-01 which represents the customer's number and univocally identifies in the CUSTOMERS' INFORMATION file all the information on the above client.

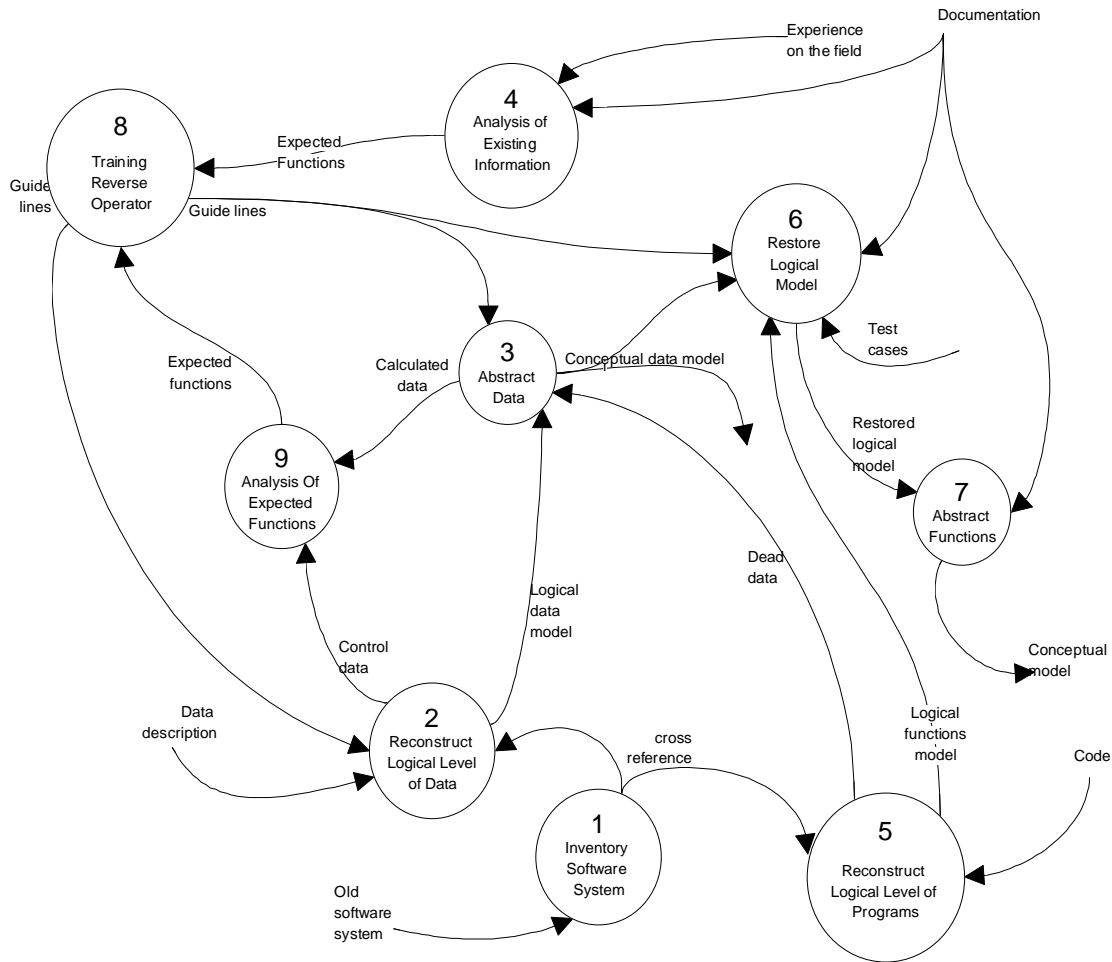
ProgramName	Module Name	mccabe	halstead	ProgramName	Module Name	mccabe	halstead	ProgramName	Module Name	mccabe	halstead
AA0000		713	141139	VA0000		516	187404	ER0000		235	67562
	AA0000	28	4832		VA0000	10	2088		ER0000	3	364
	M0001042	3	800		M0000016	1	145		M0002030	4	2380
	M0001041	5	1792		M0000017	1	1968		M0002029	5	3472
	M0001040	20	5463		M0000018	10	2504		M0002028	4	3176
	M0001039	42	9873		M0000019	20	4626		M0002027	5	2600
	M0001038	10	4617		M0000020	4	1316		M0002026	12	4048
	M0001037	16	4653		M0000021	4	1309		M0002024	1	1408
	M0001036	6	2408		M0000022	8	1800		M0002023	1	812
	M0001035	11	3616		M0000023	15	7083		M0002022	7	1848
	M0001034	7	2152		M0000024	25	5216		M0002021	5	990
	M0001033	9	3368		M0000025	7	1544		M0002020	1	726
	M0001032	22	5922		M0000026	75	38900		M0002019	2	1414
	M0001031	12	3672		M0000027	4	357		M0002018	1	708
	M0001030	14	3592		M0000028	3	252		M0002017	1	750
	M0001029	7	2240		M0000029	3	238		M0002016	1	744
	M0001028	1	1113		M0000030	3	165		M0002014	9	1386
	M0001027	28	12132		M0000031	6	1312		M0002013	18	4280
	M0001026	14	4472		M0000032	8	1224		M0002012	7	1253
	M0001025	9	5616		M0000033	7	1352		M0002011	9	2233
	M0001024	3	1694		M0000034	23	10512		M0002010	11	2030
	M0001023	6	3984		M0000035	5	448		M0002009	6	1211
	M0001022	2	1421		M0000036	13	1771		M0002008	10	2002
	M0001021	51	6872		M0000037	166	32230		M0002007	8	1393
	M0001020	8	1365		M0000038	39	6344		M0002006	29	6678
	M0001019	13	3040		M0000039	3	2688		M0002005	9	2226
	M0001018	10	3080		M0000040	4	1944		M0002004	24	5168
	M0001017	6	2128		M0000041	65	34710		M0002003	10	3744
	M0001016	4	654		M0000042	58	33410		M0002002	6	2401
	M0001015	176	31050		M0000043	55	16830		M0002001	1	220
	M0001014	15	2480								
	M0001013	28	9207								
	M0001012	3	2200								
	M0001011	44	6300								
	M0001010	22	8766								
	M0001009	4	480								
	M0001008	77	15606								
	M0001007	2	384								
	M0001006	10	1086								
	M0001005	5	642								
	M0001004	3	336								
	M0001003	3	434								
	M0001002	1	135								
	M0001001	1	2296								

**Table III. Complexity of some programs and their extracted modules**

During restoration, thanks to BACHMAN's tool [2] and VIASOFT's tool [14], and to the techniques used, the reverse operator extracts a lot of information which cannot be expressed in any of the documents produced. In particular, for many modules in the restored program, he will know not only their description but the algorithms themselves contained in the module. In fact, referring back in Figure 1, the document including the description of the behavior of the modules belongs to the flow named "restored logical model". Only a textual, not a formal, description of the aim of the module is provided for.

The data in the files formalize many design decisions taken during the past life cycle of the system, whose reasoning has been lost. As they affect the actual structure of the programs, their inadequate use prevents

the reverse operator from being able to extract some functions implemented by the code, derived from these decisions which have left no traceable reasoning. For example, the field MT02-08, which represents the total amount the client must pay in instalments to pay off the mortgage, is calculated from the mortgage amount (MT02-02), the total number of instalments (MT02-05) and the interest rate (MT02-03). The design decision has decreed that this datum be stored rather than calculated each time. The main consequence is that many modules identified during the reverse engineering process have low cohesion and high complexity and will obviously be difficult to maintain. Table III shows the complexity of sets of modules extracted from some programs, which is, in some cases, still very high. Performing test cases helps to understand very complex modules.



**Figure 4. The modified reverse engineering process**

#### 4: Modifying the process

After the first experimental period on the field, some opportune modifications were made to the process model and to the product.

To increase the reverse operator's efficiency, depending on prior knowledge of the application domain, systematic training is necessary. It is not possible to have specialized reverse operators for each domain because this would make the "system to be reversed" - "operator to be used" pairing far too rigid. We therefore decided to alter the process, as can be observed in Figure 3, inserting training activities which, by using the expected functions, explain to the operators what they should find in the programs and in the data they process. This activity is carried out by the expert in the application domain and aims to provide the operator

with guide-lines for performing the operative procedures detailed in the process. It is flexible, because the less the operator's prior knowledge of the application domain, the more exhaustive the training will be. It can be repeated as backup each time greater experience is required in the reverse operator.

It is not possible to formalize in the process the information that the tools, even when commercially known, must be carefully assessed for their efficiency, not only for their efficacy. Fortunately, in this case, the most inefficient tool was the one which derived the structure of the files, so reconstruction of the logical level of the data was modified (see Figure 2) to produce the classification of the data as conceptual, control or structural, and the description of the relationships between files.

ORIGINAL MODULE					
20296	PROCEDURE DIVISION.	20350	PERFORM RISCRIVI-ARKPT	21475	LETTURA-ARKCD-SB SECTION
20308	TAKE-ACCOUNT SECTION.	20351	IF NOT KI-0	21481	EXIT.
20309	AA.	20352	MOVE "TAKE-ACCOUNT:PT-GAR:" TO NOME-SEC	21005	LETTURA-ARKPT SECTION
20310	IF FLAGTR = 2 OR 3 OR 4 OR	20353	GO EP.	21011	EXIT.
20311	IDX-ETI-002 = 2 OR 3 OR 4 OR 5 OR 6	20360	PERFORM LETTURA-ARKPT-SB	19304	RISCRIVI-ARKPT SECTION.
20312	NEXT SENTENCE	20361	IF KI-0	19310	EXIT.
20313	ELSE	20362	GO PT-GAR.	22000	LETTURA-ARKPT-SB SECTION.
20317	GO PT GAR.	20363	GO ESCI.	22006	EXIT.
20318	PERFORM LETTURA-ARKCD.	20364	GAR.	21911	LETTURA-ARKCFG SECTION
20319	IF NOT KI-0	20369	PERFORM LETTURA-ARKFG.	21917	EXIT.
20319	MOVE "TAKE-ACCOUNT:AA:" TO NOME-SEC	20370	IF NOT KI-0	18513	RISCRIVI-ARKCFG SECTION
20320	GO EP.	20371	MOVE "TAKE-ACCOUNT:GAR:" TO NOME-SEC	18519	EXIT.
20323	PERFORM RISCRIVI-ARKCD.	20372	GO EP.	18711	LETTURA-ARKCFG-SB SECTION.
20324	IF NOT KI-0	20375	PERFORM RISCRIVI-ARKFG.	18717	EXIT.
20325	MOVE "TAKE-ACCOUNT:AA:" TO NOME-SEC	20376	IF NOT KI-0	17099	USCITA SECTION.
20326	GO EP.	20377	MOVE "TAKE-ACCOUNT:GAR:" TO NOME-SEC	17100	EP.
20332	PERFORM LETTURA-ARKCD-SB	20378	GO EP.	17101	EXIT PROGRAM
20333	IF KI-0	20387	PERFORM LETTURA-ARKGF-SB.		
20334	GO AA.	20388	IF KI-0		
20335	GOTO ESCI.	20389	GO GAR.		
20336	PT-GAR.	20390	ESCI.		
20337	IF IDX-ETI-002 = 7	20391	EXIT PROGRAM.		
20338	NEXT SENTENCE	21305	LETTURA-ARKCD-SECTION		
20339	ELSE	21311	EXIT.		
20340	GO GAR.	22310	RISCRIVI-ARKCD SECTION.		
20344	PERFORM LETTURA-ARKPT	22316	EXIT.		
20345	IF NOT KI-0				
20346	MOVE "TAKE-ACCOUNT:PT-GAR:" TO NOME-SEC				
20347	GO EP.				

**Figure 5. Original module for dynamic slicing**

The formalization of the modules is included in the structure of the logical model deliverable, whose compilation standard changes. To increase the abstraction of the information on the decisions affecting the data, it is best that the control and structural data, first, and the calculated data, successively, be analyzed by the application domain expert so that he can complete the set of expected functions, where necessary. Thus, in the process model, changes have been made to increase the communication between the data and the process reverse engineering (see Figure 4).

Finally, the experience with the test cases suggested using them to perform dynamic slicing of too complex modules, during the restoration phase. A formal description of this dynamic slicing can be found in [11].

Given a module  $M$  and supposing it contains a set of functions  $(F_1, \dots, F_n)$ ; in the set of test cases used to verify the functions of  $M$ , some subsets of test cases  $(T_1, \dots, T_n)$  must be identified, in which the generic  $T_i$  contains the equivalence classes of function  $F_i$ . Thus the test cases  $T_i$  will activate in  $M$  all and only the instructions that implement the function  $F_i$ . These instructions constitute the module  $M_i$  corresponding to the function  $F_i$ . Once the modules  $(M_1, \dots, M_n)$  corresponding to  $(F_1, \dots, F_n)$  have been extracted, a complementary module  $M_c$  to the modules  $(M_1, \dots, M_n)$  can be found in  $M$ .  $M_c$  will be the module managing  $(M_1, \dots, M_n)$ .

For example, for the module shown in Figure 5, the data for which the test cases have been constructed are:

$IDX-ETI-002 : \{1, \dots, 20\}$

*Index of types of transaction table*

$FLAGTR-WS : \{0, \dots, 9\}$

*Work area flag whose value is paired with the index of types of transaction table.*

The combined values of these two variables specify the exact nature of the transaction the client intends to make with the bank.

$KI-0 : \{\text{true}, \text{false}\}$

*Control variable which states the result of access to the file. If true, access exists, if false, then access to the file was not possible.*

The value of this variable is tested (in the cases in Figure 6) only on exit from PERFORM, which enables reading of the files.

Thus the original module was substituted by the structure shown in Figure 7.

The complexity is modified as shown in Table IV. This is formalized in the reverse engineering process model by modifying the procedure corresponding to the activity "reconstruct logical level of programs". In fact for all modules exceeding a predefined level of complexity, dynamic slicing will be performed using the test cases relative to the module in question.



1 <sup>a</sup> EQUIVALENCE CLASS		2 <sup>a</sup> EQUIVALENCE CLASS		3 <sup>a</sup> EQUIVALENCE CLASS	
FLAGTR-WS in (2,3,4)		FLAGTR-WS in (2,3,4)		FLAGTR-WS NOT in (2,3,4)	
IDX-ETI-002 in (2,3,4,5,6)		IDX-ETI-002 in (2,3,4,5,6)		IDX-ETI-002 NOT in (2,3,4,5,6,7)	
KI-0=FALSE		KI-0=FALSE		KI-0=FALSE	
KI-0=TRUE		KI-0=TRUE		KI-0=TRUE	
MODULO TAKE-ACCOUNT-A/C-DEPOSIT-SAVINGS		MODULO TAKE-ACCOUNT-INVESTMENT-PORTFOLIO		MODULO TAKE-ACCOUNT-SURETIES	
PROCEDURE DIVISION 20306 AA. 20314 MOVE ZERO TO KEY-CD.		PROCEDURE DIVISION 20336 PT-GAR. 20341 MOVE ZERO TO KEY-PT.		PROCEDURE DIVISION. 20364 GAR. 20365 MOVE ZERO TO KEY-CFG.	
20317 PERFORM LETTURA-ARKCD. 20318 IF NOT KI-0 20319 MOVE TAKE-ACCOUNT-AA:TONOME-SEC 20320 GO EP.		20344 PERFORM LETTURA-ARKPT 20345 IF NOT KI-0 20346 MOVE TAKE-ACCOUNT-PT-GAR:TONOME-SEC 20347 GO EP.		20369 PERFORM LETTURA-ARKCFG. 20370 IF NOT KI-0 20371 MOVE TAKE-ACCOUNT-GAR:TONOME-SEC 20372 GO EP.	
20323 PERFORM RISCRIVI-ARKCD. 20324 IF NOT KI-0 20326 MOVE TAKE-ACCOUNT-AA:TONOME-SEC 20326 GO EP.		20350 PERFORM RISCRIVI-ARKPT 20351 IF NOT KI-0 20352 MOVE TAKE-ACCOUNT-PT-GAR:TONOME-SEC 20353 GO EP.		20375 PERFORM RISCRIVI-ARKCFG. 20376 IF NOT KI-0 20377 MOVE TAKE-ACCOUNT-GAR:TONOME-SEC 20378 GO EP.	
20332 PERFORM LETTURA-ARKCD-SB 20333 IF KI-0 20334 GO AA. 20335 GO TO ESCI. 20390 ESCI. 20391 EXIT PROGRAM. 21305 LETTURA-ARKCD SECTION 21311 EXIT 22310 RISCRIVI-ARKCD SECTION. 22316 EXIT. 21475 LETTURA-ARKCD-SB SECTION 21481 EXIT. 17099 USCITA SECTION. 17100 EP. 17101 EXIT PROGRAM.		20380 PERFORM LETTURA-ARKPT-SB 20361 IF KI-0 20362 GO PT-GAR. 20363 GO ESCI. 20390 ESCI. 20391 EXIT PROGRAM. 21005 LETTURA-ARKPT SECTION 21011 EXIT. 19304 RISCRIVI-ARKPT SECTION 19310 EXIT. 22000 LETTURA-ARKPT-SB SECTION 22006 EXIT. 17099 USCITA SECTION. 17100 EP. 17101 EXIT PROGRAM.		20387 PERFORM LETTURA-ARKCFG-SB 20388 IF KI-0 20389 GO GAR 20390 ESCI. 20391 EXIT PROGRAM. 21911 LETTURA-ARKCFG SECTION 21917 EXIT. 18413 RISCRIVI-ARKCFG SECTION 18519 EXIT. 18711 LETTURA-ARKCFG-SB SECTION. 18717 EXIT. 17099 USCITA SECTION. 17100 EP. 17101 EXIT PROGRAM.	
COMPLEMENT					
MODULO COMPLEMENTARE AI CASI 1,2 E3 PROCEDURE DIVISION. 20296 TAKE-ACCOUNT SECTION. 20308 AA. 20309 IF FLAGTR-WS =2 OR 3 OR 4 OR 20310 IDX-ETI-002 =2 OR 3 OR 4 OR 5 OR 6 ..... CALL TAKE-ACCOUNT-A/C-DEPOSIT-SAVINGS		20312 ELSE 20313 GO PT-GAR. 20335 GO TO ESCI. 20336 PT-GAR. 20337 IF IDX-ETI-002=7 ..... CALL TAKE-ACCOUNT-INVESTMENT-PORTFOLIO		20339 ELSE 20340 GO GAR. 20363 GO ESCI. 20364 GAR. ..... CALL TAKE-ACCOUNT-SURETIES. 20390 ESCI. 20391 EXIT PROGRAM.	

Figure 6. Results of dynamic slicing

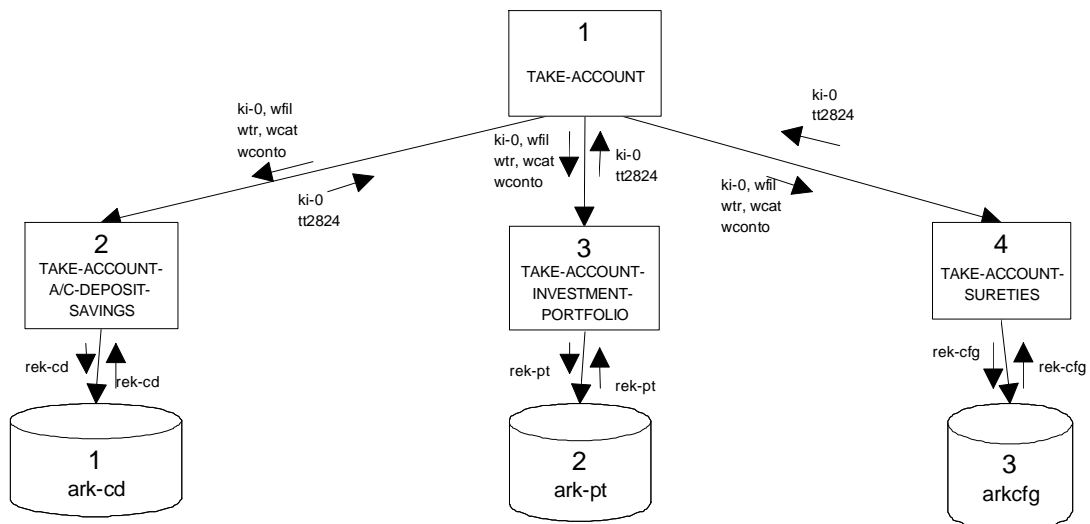


Figure 7. Program structure after dynamic slicing

Program/Module	Halstead	McCabe
Original program	3276	12
TAKE-ACCOUNT-MODULE	1512	3
TAKE-ACCOUNT-A/C-DEPOSIT-SAVINGS-MODULE	1920	4
TAKE-ACCOUNT-INVESTMENT-PORTFOLIO-MODULE	1873	4
TAKE-ACCOUNT-SURETIES-MODULE	1813	4

**Table IV. Module complexity before and after extraction**

## 5: Conclusions

This paper describes the results of experimentation on the field of a reverse engineering process and the improvements made on the basis of the data obtained.

The feedback gained from the quality of the product and the quality of the process is particularly enlightening. In this case, the usability of the documentation to understand the programs better suggested some improvements to the process. This feedback enabled closer connection to be made in the process model between the static and the dynamic information which can be extracted from the application domain and the user context of programs.

In addition, a synergetic analysis in the process of learning about the programs was defined. The programs to be documented were so complex that dynamic analysis alone would have been inadequate: in fact, in the original programs, each test case would have activated so many instructions that a great deal of man time would have been required to understand them. Instead, using prior static analysis, the programs were decomposed and classified essentially into two types: modules whose aim and behavior are clear, and modules whose aim is clear but behavior is not. The latter, more complex modules, can be understood better after dynamic analysis with the test cases used to verify the results of the reverse engineering process.

Unfortunately, the modified process has been operative for too short a time for data to be available to assess its quality after the modifications.

Finally, the experience highlighted the need to improve testing and validation of the functionality and efficacy of the tools used. Even well known tools which have been on the market some time may be inadequate because, for the most part, they have been used in pilot projects rather than effective production. Hence, the reverberations of their actions are still unknown.

## References

- [1] F. Abbattista, F. Lanubile, and G. Visaggio, "Recovering conceptual data models is human intensive", Fifth International Conference on Software engineering and Knowledge engineering, San Francisco, California, 1993.
- [2] BACHMAN, *Bachman/Analyst Reference Manual*, Release 4.10.
- [3] G. Canfora, A. Cimitile, and U. De Carlini, "A logic based approach to reverse engineering tool production", *Conference on Software Maintenance*, Sorrento, Italy, 1991.
- [4] G. Canfora, A. Cimitile, and M. Munro "RE<sup>2</sup>: reverse engineering and reuse re-engineering", *Computer Science Technical Report 8/92*, University of Durham, School of Engineering and Computer Science, 1992.
- [5] E. J. Chikofsky, and J. H. Cross II, "Reverse engineering and design recovery: a taxonomy", *IEEE Software*, January 1990.
- [6] G. Como, F. Lanubile, and G. Visaggio, "Design recovery of a data-strong application", *3rd International Conference on Software engineering and Knowledge engineering*, Illinois, USA, 1991.
- [7] F. Cutillo, F. Lanubile, and G. Visaggio, "Extracting application domain functions from old code: a real experience", *2nd Workshop on Program Comprehension*, Capri, Italy, 1993.
- [8] F. Cutillo, P. Fiore, and G. Visaggio, "Identification and extraction of domain independent components in large programs", *Working Conference on Reverse Engineering*, Baltimore, 1993.
- [9] F. Cutillo, F. Lanubile, and G. Visaggio, "Using program slicing for software comprehension", *IEEE Workshop Notes on Software Comprehension*, Orlando, Florida, 1992.
- [10] FORMATICA, "Definition of the production line. reverse engineering: process model", *Int. doc. no. 69*. September 1992
- [11] R. Gopal, "Dynamic program slicing based on dependence relationships", *Conference on Software Maintenance*, Sorrento, Italy, 1991.
- [12] F. Lanubile, and G. Visaggio, "Function recovery based on program slicing", *Conference on Software Maintenance*, Montreal, Quebec, Canada, 1993.
- [13] M. Weiser, "Program Slicing", *IEEE Transactions on Software Engineering*, vol. SE-10, n°4, July 1984.
- [14] VIASOFT, *VIA/Renaissance User Manual*, *VIA/Insight User Manual*, *VIA/SmartDoc User Manual*.