Iterative Reengineering of Legacy Functions

Alessandro Bianchi*, Danilo Caivano*, Vittorio Marengo°, Giuseppe Visaggio*
** Dipartimento di Informatica – Università di Bari - Via Orabona, 4, 70126 Bari – Italy*
{bianchi, caivano, visaggio}@di.uniba.it
° Dipartimento di Statistica – Università di Bari – Via Rosalba, 53 70124 Bari - Italy
vmarengo@dss.uniba.it

Abstract

This paper describes a process of gradual reengineering of the procedural components of a legacy system. The process is integrated and completed by the data reengineering process analyzed in a previous paper by the same authors. The proposed method enables the legacy system to be gradually emptied into the reengineered system, without needing to either duplicate the legacy system or freeze it. The process consists of evolving the legacy system components toward firstly a restored system and then toward the reengineered system. Meanwhile, the legacy system can coexist with both the restored and the reengineered parts. By the end of the process, a single system will be in existence: the reengineered one. The method has been applied to reengineer a real system and demonstrated its ability to: support gradual reengineering, maintain the system at work during the process, minimize the need to freeze maintenance requests, renew the operative environment of the reengineered system with respect to the legacy system and, finally, eliminate all the system's aging symptoms.

1. Introduction

Our research deals with the problems of reengineering legacy systems. The ample volume of literature in this field testifies to the interest it arouses in both the scientific and the industrial communities: the works by Sneed ([Sne95] and [Sne96]), Coyle ([Coy00a] and [Coy00b]), Quilici ([Qui95]), Robertson ([Rob97]) and others ([BLW99], [HHL97]) are just a few examples. The reason why this issue is so important is because an enormous number of working applications were developed using methodologies and programming languages that are now out of date. Just the systems written in Cobol have been estimated by Coyle [Coy00b], to account for more than 100 billion LOC.

These systems and the data they process are vital assets for the companies that use them. However, over the years they have been subjected to countless maintenance processes that inevitably caused progressive aging [Vis97]. This degeneration, together with the continual and often unexpected evolutions of the applicative domain that arise, leads to a high number of requests for maintenance that take longer and longer to satisfy as the system continues to age. Reengineering is necessary, indeed indispensable, to overcome many of the most serious aging symptoms [Vis97]. In addition to improving the quality of the system, the reengineering process should enable new functions to be introduced and new technologies to be adopted, to ensure efficient management of the information container in the legacy system, as shown in [NNB98], and [Rob97].

The reengineering process is an intrusive one because it requires the data and the procedures to be restructured all at the same time. The legacy system obviously cannot stop working during the process. However, during reengineering of a procedure, the latter cannot be modified otherwise a loop would be created between the maintenance and reengineering processes that could end in an undesirable interaction. For this reason, the reengineering process has to be done on few procedures at a time and to last as short as possible, so that only requests for change having an impact on the few procedures currently being reengineered need to be frozen. The reengineered system components will coexist with all the others. Finally, the maintenance activities, if required, can be carried out on both the reengineered and legacy components, depending on the impacted procedures.

Our approach to reengineering has been organized in such a way as to satisfy all these requirements. In a previous work, [BCV00], we dealt with reengineering the data in an aged legacy system; in the present paper we shall look at the issues involved in reengineering the functions. The method proposed has been experimented on a part of a legacy system called Fa2000, and we shall refer to the reengineering experience gained with this system to provide examples of the concepts introduced.

The paper has been organized as follows: Section 2 analyzes the other main approaches described in the literature and points out the innovative aspects of our approach. Section 3 describes the architecture that the proposed process will impart to the legacy system and the restored system, while they coexist. Section 4 analyzes the gradual reengineering process we propose. Section 5

outlines the case study on which the method was experimented. Finally, in Section 6 the main conclusions are drawn.

2. Related Works

The reengineering techniques available in literature [Big89], [Bro93], [CC90] establish that a reengineering process should involve the entire software system. For this reason, the software system must be frozen until the execution of the process has been completed; in other words, no changes are possible during its execution. In fact, if a change or a system evolution were introduced, the legacy system and the renewal candidate would be incompatible, and the software engineers would have to start the process all over again from the beginning. This situation causes a loop between the maintenance process and the reengineering process.

To overcome this problem, several authors have suggested wrapping the legacy system, and considering it as a black-box component to be reengineered. In this way, it is possible just to add new functions and a new, more useful interface with relative ease [Sne96]. However, this technique does not solve the problem of inertia of the aged legacy system, which will remain unchanged. For this reason, if the wrapped system needs to be evolved in some way, all the consequences of the aging symptoms indicated in [Vis97] will re-emerge. Therefore, although the wrapping approach offers a solution to the problem of coexistence between the aged programs and the new ones requiring relatively little effort, it does nothing to solve the problem of maintenance of aged programs.

The most similar approaches to the one described here are the *Chicken Little Strategy* [BS95] and the *Butterfly Methodology* [WLB97]. These approaches, like the one we propose, are based on the assumption that *data in a legacy system are logically the most important part of the system* and that *from the viewpoint of development of the target system, it is not the ever-changing legacy data that is crucial, but rather its semantics or schema(s)* [WLB97].

The Chicken Little strategy gradually rebuilds the legacy system on the target platform using modern tools and technology, by applying an 11-step process very similar to the one proposed below. During migration, the legacy and target system form a composite information system in which the various components access data through a mediating module known as a Gateway. This solution requires legacy data to be duplicated and reengineered in the target system: for each data access, therefore, both databases need to be accessed. However, many of the complex features found in modern databases, such as integrity, consistency constraints and triggers may not exist in the legacy database and hence cannot be exploited by new applications [BLW 99], so it is hard to update the system's consistency.

In the Butterfly Methodology, both the legacy and the target data system can continue to operate, but without sharing functions or data. The Butterfly methodology thus focuses on legacy data migration and develops the target system as an entirely separate process. Its main weakness with respect to data migration is the need to freeze the old database and to use it only for reading, while the changes are kept in a temporary auxiliary store, so that data access time increases unacceptably. Moreover, this methodology does not allow the legacy functions to coexist with the reengineered and newly constructed functions [BLW99].

The advantage of our method is that it enables coexistence of data and functional components of the legacy system and the ones in the reengineered system. The data structure is reengineered, rather than simply migrated as in the competitor approaches. In practice, the legacy data are translated into the new database without duplication [BCV00]. The legacy database does not have to be duplicated nor frozen but is gradually transferred into the new database, which can be based on a more modern database management system. The components of the reengineered system coexist with those of the legacy system and use either the legacy or the new database, depending on where the data to be processed are stored. The components of the legacy system will gradually be reengineered and the old system will finally disappear. By the time the legacy system has been completely reengineered, the old database will also have been completely emptied.

3. Architecture of the System

The iterative reengineering method is based on gradual evolution of a legacy system over a period of time, by operating on the various components in sequence. It is based on the rationale described in [Vis97] and in [BCV00]. The basic concepts are briefly outlined below, for the sake of clarity, but readers should refer to the those works for the full explanation.

3.1. Basic Concepts

The proposed method enables iterative reengineering of a legacy system, while guaranteeing coexistence among the various components that will go through various different states during the process. Each component may be in one, and only one, of the following states during the various stages of the reengineering process:

- *legacy*, i.e. components of the legacy system that have not yet been reengineered;
- *restored*, i.e. functions with the same structure they had in the legacy system, but that access data through the new data banker, that alone recognizes the physical structure of the database;

- *reengineered*, i.e. components of the legacy system that have already been modified, and whose quality has reached the desired levels;
- *new*, i.e. components that did not exist in the legacy system, but have been added to introduce new functions in the same application domain.

For sake of completeness, from a conceptual point of view, the legacy data are partitioned into two classes:

1. *essential data,* needed to carry out the application's business functions;
2. *residual data*, that are not necessary for carrying out the business functions, but are used by the legacy system and must therefore remain in the database until the procedures that use them have been reengineered.

The *essential data* contain:

- *conceptual data*, specific to the application domain, and for which they have a precise conceptual meaning;
- *necessary structural data*, that do not belong to the above class but constitute the primary keys to the tables making up the database.

Instead, the *residual data* class contains all the data that existed in the legacy database, and which should ultimately be eliminated to improve the software quality. They are classified as:

- *control data*, that communicate to a procedure that an event occurred during execution of other procedures, thus controlling the behavior of the former one;
- *redundant structural data*, used to organize and support the data structures of the legacy system, but which are not strictly required; with a better design of the database they can be removed;
- *semantically redundant data*, whose definition domain is the same as, or is contained in, the definition domain of other data, while each equal value in the two definition domains is interpreted in the same way;
- *computationally redundant data* which can be computed starting from a different set of data included in the same database.

3.2. The Reengineered System Architecture while Reengineering

The method proposed in the present work is based on the architecture shown in figure 1.

According to the service required, the user can access original, restored, reengineered or new functions through the "USER INTERFACE".

The package labeled "LEGACY COMPONENTS" in figure 1 represents the aged system that operates on a set of data recorded in the database, labeled "LEGACY DB". As stated above, the Legacy DB contains both essential and residual data. The component in fig. 1 labeled "NEW DB" indicates that part of the system that will have taken over recording essential data from the Legacy DB by the end of the reengineering process. In addition, all essential data referring to the new functions added to the system during reengineering will be recorded in the New DB.

The structure of the New DB has been organized according to the modern approach, introducing new functions that will enable more efficient management of the database through an updated database management system (the package labeled "DB MANAGER" in Fig.1).

The residual data of the Legacy DB will be migrated during the process into the component in fig. 1 labeled "RESIDUAL DB". It is managed by the "RESIDUAL DATA MANAGER", that may be either the new database management system or that of the legacy software. The "METADATA" package contains all the data needed to identify the physical location of a field in the database.

The "DATA BANKER" receives all the requests for data from each procedural component. Through the METADATA, it can understand where the data needed to satisfy the request are stored, and will ask the DB MANAGER for the data in the NEW DB and the "DATA LOCATOR" for those stored in the RESIDUAL DB.

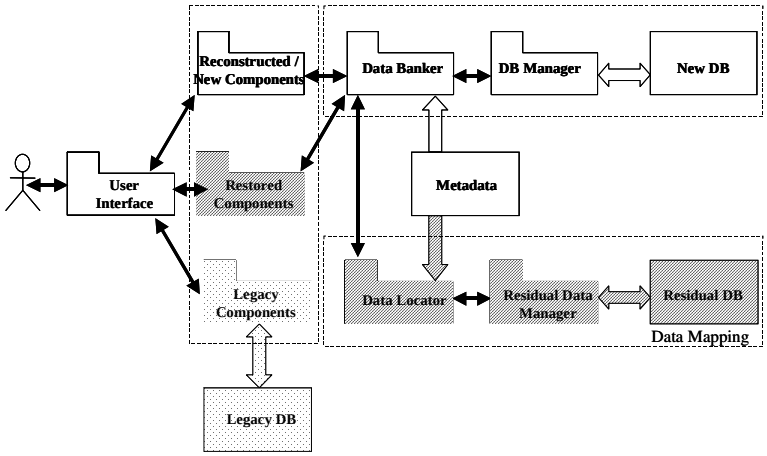


Figure 1. The reengineered system architecture while reengineering.

After receiving the request from the DATA BANKER , the DATA LOCATOR will discover from the METADATA where the data required are physically located, and formulate the request for access to the RESIDUAL DATA MANAGER.

It should be noted that the various architectural components in Figure 1 are shown with three different degrees of shading, representing three different life spans. More precisely, the components with light shading ("LEGACY COMPONENTS" and "LEGACY DB") are the original components destined to disappear gradually as the process goes on; those with darker shading are transition components that allow the procedures to be reengineered after the data; the components with no shading are those that will remain at the end of the process. It is worth noting that both DATABANKER and METADATA packages will remain after reengineering. METADATA contains the physical location (what position in what table in NEW DB) for each datum accessed by applicative functions. This implies that if the physical structure of the NEW DB changes only data in METADATA need to be changed.

4. The Process

The reengineering process is shown in figure 2, the various phases each being enclosed in a box, while the label on the arrow linking two successive phases indicates the output produced by the relative phase. The various phases will be analyzed below, concentrating on the parts that reengineer the functions, while the data reengineering aspects are only touched on very briefly, and readers should refer to [BCV00] for full details.

4.1. Analyze the legacy system

To understand this phase, it should be remembered that during reengineering of a legacy system component, all the requests for change that have an impact on this component must be put on hold until the component is reengineered. For this reason, the system is partitioned into components with no inter-dependencies, i.e., there are no client-supplier relationships between components. In this way, the reengineering of a component is independent from the reengineering of the other. This partitioning has the aim of identifying the components in the legacy system that are suitable for reengineering, so minimizing the time that requests for maintenance have to be kept waiting.

In this sense we express the *mean time to maintenance request* for the i-th component ($MTTMR_i$) with the following formula: $MTTMR_i = \sum_{j=1}^{n_i} \frac{\Delta t_{j,i}}{n_i}$, where:

- n_i is the total number of requests for maintenance made for the i-th component;
- $\Delta t_{j,i} = t_{j,i} - t_{j-1,i}$ is the time interval between two successive requests for maintenance of the i-th component.

The values for n_i and $\Delta t_{j,i}$ are obtained from the historical archives of the system. When the expected time for reengineering the i-th component, RT_i, is less than $MTTMR_i$, then it is reasonable to suppose that there are unlikely to be requests for maintenance of this component during the time it is being reengineered. Vice versa, if RT_i, is higher than $MTTMR_i$, then the component must be subdivided into several subcomponents, to prevent the requests for maintenance that are likely to be made during this time from being held up for too long time.

The preliminary phase of analysis of the legacy system also involves identification and analysis of the usage relationships between the data files and the various components, so as to be able to establish how best to set about the reengineering of the single components.

The output produced by the legacy system analysis phase ("SET OF COMPONENTS" in figure 2) is a description of the set of components that can be reengineered in succession, indicating the various reengineering priorities, according to MTTMR values and requirement priorities.

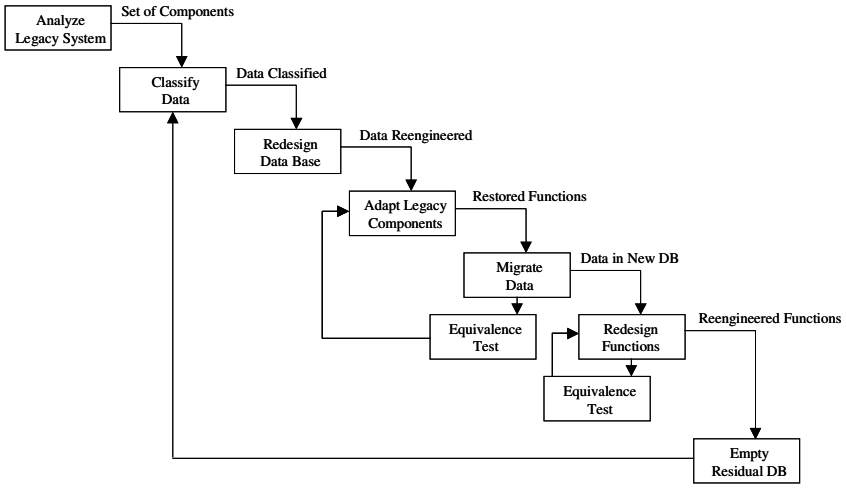


Figure 2. Iterative reengineering process.

Table 1 shows an excerpt from this description, where each program is associated with the functions it has and the set of files it manages, and its access mode is indicated: Creation (C), Reading (R), Updating (U) Deletion (D).

Component	Function description	Data file accessed	Priority
FATMPB.CBL	Management of supplier archives	ARCCOD (R), ARCFED (R), DAT080 (R), DAT240 (R), DATFAR (R), DATFED (R), DATFOR (R)	HIGH
FDITTB.CBL	Display supplier archives	DAT080 (R), DATFAR (R), DATFED (R), DATFOR (CRUD), PARSTA (R)	HIGH
FPNUMB.CBL	Calculation of progressive numbers	DATPRO (CRUD), DATPOS (CRUD)	LOW
FTABEB.CBL	Official drug tables	ARCFED (CRUD), DATFAR (R), DATFED (CRUD), DAT080 (CRUD), DAT240 (CRUD)	MEDIUM
SUBSSN.CBL	Generation of statistics for the national health service	ARCCOD (R), ARCFED (R), DATCAT (R), DATFAR (R)	LOW
...	...	...	...

Table 1. An excerpt from the description obtained with the data classification phase

In the experiment, this phase was carried out with the support of the MicroFocus Revolve tool, version 5.0 [Mer00]. In particular, the tool was used to establish the usage relationships between data files and the programs.

4.2. Classify data

This phase involves identification, and interpretation of the data recorded in the Legacy DB, and of their reciprocal relationships. During this phase the data are also classified according to the definitions given in section 3.1. At the end of the data classification phase a table is obtained that records all the non duplicated data present in the Legacy DB. This phase was also carried out in the experiment with the support of the MicroFocus Revolve 5.0 [Mer00].

4.3. Redesign Database

During the Redesign Database phase the various data classified in the previous phase must be restructured so as to reorganize them in a more effective, efficient way inside the New DB. In fact, in an aged legacy database the database is often not normalized at all or not adequately normalized. This phase therefore aims to establish the structure of the New DB, applying effective database design techniques. In our case, the normalization technique described in [Smi85] was used, and the New DB is organized in a relational structure.

4.4. Adapt Legacy Components

The following phase, Adapt Legacy Components, aims to render the legacy system programs compatible with the data to be reengineered. For this purpose, within each legacy system program that accesses the data to be reengineered, all the instructions involved in accessing the data must be identified. These instructions must then be replaced by new ones that, instead of accessing the data directly, call for this service from the data banker: it will be the component accessing data on behalf of the calling program.

Legacy Program

```
.
.
LETTURA-DATFOR-DIRETTA.
    READ DATFOR INVALID GO PRENDI-DITTA-ALFABETICA.
    IF LCK-R            GO LETTURA-DATFOR-DIRETTA.
    MOVE X4             TO SALDO.
    GO DITTA-SELEZIONATA.
:
```

a

Adapted Program

```
.
.
LETTURA-DATFOR-DIRETTA.
    CALL "READ" ID-DATFOR DATFOR LEGACY-FILE-STATUS.
    EVALUATE LEGACY-FILE-STATUS
        WHEN 00 GO PRENDI-DITTA-ALFABETICA.
        WHEN 99 GO LETTURA-DATFOR-DIRETTA.
    END-EVALUATE
    MOVE X4             TO SALDO.
    GO DITTA-SELEZIONATA.
:
```

b

Reengineered Program

```
.
.
LETTURA-DATFOR-DIRETTA.
    CALL "READ" ID-DATFOR DATFOR LEGACY-FILE-STATUS.
    EVALUATE LEGACY-FILE-STATUS
        WHEN 00 GO PRENDI-DITTA-ALFABETICA.
        WHEN 99 GO LETTURA-DATFOR-DIRETTA.
    END-EVALUATE
    SUBTRACT AVERE FROM DARE GIVING SALDO.
    GO DITTA-SELEZIONATA.
:
```

c

Figure 3. An excerpt of code from a) the legacy program, b) the adapted program and c) the reengineered program

Figure 3 shows an example of adaptation of a legacy component: in Fig.3.a, there is the piece of code belonging to the legacy system program and in Fig.3.b, the corresponding piece of code after adaptation. In the former there is a "Read" instruction which retrieves data from the legacy datafile DATFOR. This instruction is replaced in the latter by the service request to the data banker. The service request is expressed by a call to a new program named READ, to which the parameters ID-DATFOR (the identifier of the firm), DATFOR (the file to be read) and LEGACY-FILE-STATUS (a return parameter indicating the current status of the legacy file) are passed.

After completion of this phase, the system can operate in two different ways, depending on the user request:

1. execute functions in the legacy system that operate on the legacy database, if the request does not involve reengineered data;
2. otherwise, execute functions with Restored Components that operate on reengineered data.

Thanks to the Data Banker, this dual possible behavior of the system is transparent to the users, who will continue to operate as they did formerly with the legacy system. However, if they access reengineered data they will in fact be referring to those recorded in the New DB or the Residual DB rather than those in the Legacy DB. During the experimentation of the method, the instructions for accessing the data were identified with the support of the MicroFocus Revolve 5.0 tool [Mer00], while the programs were updated using the development environment Acucobol, version 4.3 [Acu00].

4.5. Migrate Data

The data in the legacy database are progressively migrated to the new database; instead, data identified as not essential are migrated to the residual database. In the experiment, the Data Migrator tool, developed ad hoc in the Software Engineering Research LABoratory (SER_Lab) of Bari University, was used. For each data file being reengineered, it reads all the data it contains and, according to the information contained in the Metadata, copies them into the Residual DB or New DB.

4.6. Redesign Functions

In the Redesign Functions phase, the reengineering of the various functions is carried out, causing them to evolve from the r*estored* to the *reengineered* state. During this phase the software engineer analyzes the quality deficiencies of each procedure and introduces suitable remedies. More precisely, the software engineer:

- restructures the components to bring their quality up to the desired standards; particular attention has to be paid to information hiding and to all the features that can help to make software maintenance easier;
- individuates any procedures among those composing the component being reengineered that are clones of procedures already present in the reengineered system, and eliminates such clones in favor of the best quality procedure;
- updates data access management to match the new organization;
- improves the algorithms used;
- updates the modules interface;
- updates the user interface;
- executes the maintenance operations that had been put on hold during the reengineering process;
- updates the programming language to more modern versions.

Each of the above operations must be carried out by reusing as far as possible the components originally present in the legacy system. This strategy is prompted not only by reasons of economy but also in the interests of preserving the skills the maintainers have developed while operating on the system. It is assumed that the maintenance team that operated on the legacy system is also likely to operate on the reengineered system, and that it is therefore desirable to preserve such familiarity with the system as is compatible with the updating process.

The most salient operations are analyzed below.

Data access management

While a procedure is being reengineered, the decisions about how to represent information in the data or procedures made for the legacy system can be revised, in order to eliminate the redundant computational data as far as possible. For example, figure 3.b shows the datum X4 in DATFOR that expresses the SALDO (balance). X4 is not essential and is therefore in the Residual DB. Because the items DARE (amount owing) and AVERE (amount due) are included in DATFOR, the procedure has been transformed into the one shown in figure 3.c. In this case, all the values for SALDO (balance) have been cancelled from the Residual DB and the SALDO field has also been eliminated from the Metadata. From now on, when DATFOR is read, the Data Banker will not be asked to specify the SALDO (balance).

Figure 4. The initial screen of Fa2000.

Updating of the user interface

Many aged legacy systems present the results with a character-based interface. For example, figure 4 shows the first screen of the Fa2000 system. As can be seen, it is in MS-DOS and features a character-based interface, with Italian text messages. Navigation in the legacy version is by means of Function Keys. Figure 5 shows another screen display of the legacy system, that enables the data to be displayed and modified (by means of function key F9), printed (F10) or a supplier's record to be eliminated (with the DEL key). Each screen presents only a subset of data relating to a given supplier: to display the other data press the relative Function Key (e.g. F1 to see the 1st page, F3 the 3rd, ...). User interaction with systems with this kind of interface is generally very complex [PRS94] and can give rise to various kinds of error. It is therefore best to update the interface and system interaction modes during the reengineering process.

In our Fa2000 system, the user interaction mode was updated during reengineering, being changed to the Windows-like approach depicted in figure 6, that shows the screen that appears in the reengineered version of the display in figure 5.

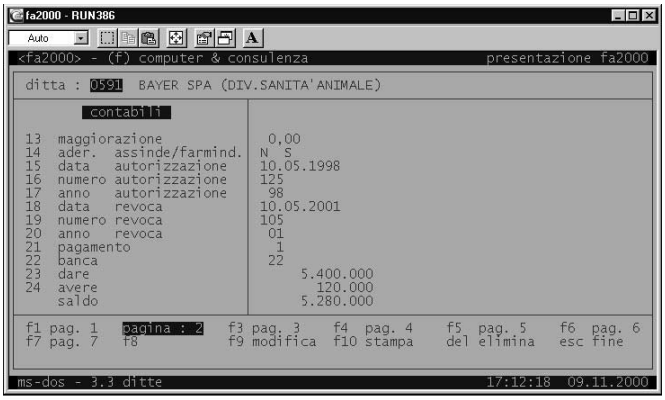


Figure 5. A display screen of the original management of a supplier's data.

Execution of maintenance

The highly dynamic nature of the application domains of aged legacy systems gives rise to a great number of requests for maintenance. During reengineering, the system manager must therefore expect to dedicate a large part of the effort to maintenance operations.

In our experiment, one of the maintenance operations required on the system was currency management. In fact, in the legacy system the amounts are expressed in Italian liras, as shown in figure 5. It was therefore necessary to update the system to express the dual currency regime that will become the norm in Europe in the year 2002. The maintenance operation carried out enabled the amounts to be expressed in the dual currency, Italian liras and Euros, as shown in figure 6.

Porting the development environment

Another characteristic of aged legacy systems is that they have usually been developed in environments and programming languages that are now obsolete. During reengineering it is possible to change to more modern environments and languages that offer more ample functions.

In the case of Fa2000 system, we used a more evolved programming environment than the one used originally, so that we adopted Acucobol version 4.3, instead of the version 2.4 by which the previous version of the system was developed. Thanks to this porting, a large part of the code could be reused, as the programming language characteristics of the legacy version were much the same as those of the more modern version. However, the version 4.3 offers greater opportunities for managing both the user interface in Windows-like mode and data access through an SQL-Server.

4.7. Equivalence Test

Fig.2 shows two testing phases of the activities carried out, run straight after the Migrate Data and Redesign Functions phases. In both cases the aim was to verify that the new system had the same functions as the legacy system. Thus the equivalence tests aim to compare the behavior of the reengineered system with that of the legacy system.

4.8. Empty Residual DataBase

The examples given in section 4.6 show how some of the accesses by the Restored Components to the Residual DB no longer occur when the functions have been reengineered. In this case, as the residual data are used *only* by functions that have now been reengineered, they are no longer necessary and can be eliminated from the Residual DB.

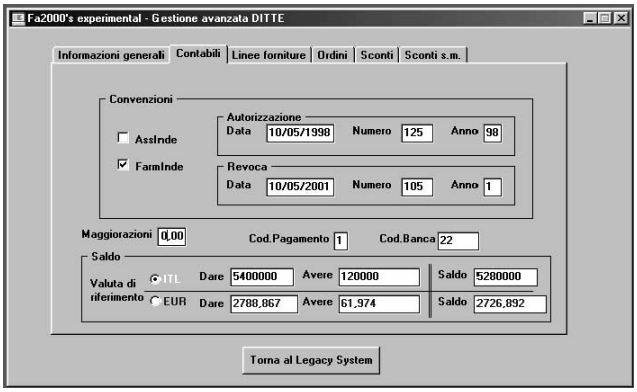


Figure 6. A display screen of the reengineered management of a supplier's data

Note that the Residual DB is not necessarily emptied after the end of each loop, in that some residual data could be accessed by functions not yet reengineered. Surely the Residual DB will be empty at the end of the reengineering process, as no functions will access its data. For this reason, the components Residual Data Manager and Data Locator can also be eliminated from the system, as they will have become superfluous.

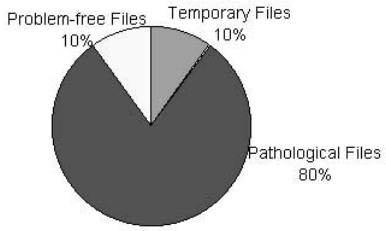


Figure 7. Summary of the features of the files in Fa2000

4.9. Iteration

Once a given set of components of the legacy system has been reengineered, the process is repeated, applying it to the next set of components, until the whole legacy system has been reengineered.

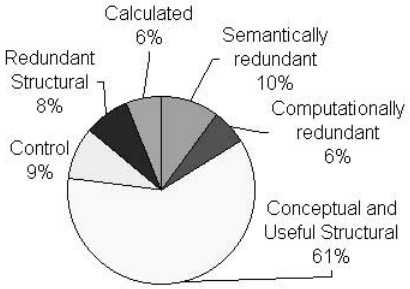


Figure 8. Summary of features of data in Fa2000

5. Case Study

The described method has been experimentally applied to reengineer an aged legacy system: Fa2000. It supports supplier management in about 100 chemists' shops in Italy. The data managed by Fa2000 relate to the suppliers and pharmaceutical products sold in the chemists' shops, the chemical and pharmaceutical properties of the various products, and the health, legal and economic issues involved. The system was developed as from 1987 and the first version was distributed starting from 1989. Various maintenance operations, of corrective, adaptive and perfecting types were carried out by the software house that had produced the first version. The version used in the experiment had been released in December 1999.

The system underwent a great number of maintenance operations during its life-span, and these contributed to degrade its quality. Despite the many operations carried out, the system still features many unstable components due to the dynamic evolution of the application domain.

Reengineering involved about the 10% of the system, for a total effort of about 230 man /hours, to reengineer the procedures. Over 6 months, about 32 requests for maintenance were made. A total of 5 changes had to be held up for a mean time of 10 working days.

Figures 7 and 8 summarizes the results of the analysis of legacy system and classification of legacy data. Note that only 10% of the files were problem-free, while as many as 80% were pathological, i.e., their content was changed by more than one program [Vis97]. The symptom of high coupling among modules is very evident. Moreover, only the 61% of data were essential data, whereas 39% were residual.

After data reengineering [BCV00], all the files were problem-free, while as regards the data, both redundancy and calculated data were completely removed, the quantity of control data decreased to 2% and the essential data amounted to 98% (1.4% of necessary structural data and 96.6% conceptual data). Just to give an idea of the task undertaken, the overall number of fields was equal to 8000 before and 6326 after reengineering.

Metric	Legacy System	Reengineered System
Number of Modules	729	201
Cyclomatic Complexity	805	284
Mean Cyclomatic Complexity	2.102	1.706
LOC	7,442	1,046

Table 2. Values of the complexity metrics for Fa2000 before and after reengineering.

Apart from improving the system aspects described above, reengineering improved its complexity. Table 2, summarizes the values for the metrics:

- *Number of Modules*, expressed as the sum of Cobol paragraphs, sections and external routines;
- *Cyclomatic Complexity*, calculated on the whole system;
- *Mean Cyclomatic Complexity*, calculated as the mean value for each module;
- *Lines of Code* in the whole system

before and after reengineering, calculated with the MicroFocus Revolve 5.0 tool.

The reduction in the number of modules from 729 to 201 and the corresponding reduction in lines of code:

- are due to different management of the interface, that was carried out by specific procedures in the legacy system whereas in the reengineered system it is carried out by components belonging to the new environment (40% of reductions);
- are due to the elimination of clones (35%);
- are due to the reduction of managed data (25%).

Although Cyclomatic Complexity already had a good value for the legacy system, it improved further. The good state even before reengineering was due to the characteristics of the system. It is strongly data-oriented and makes calculations requiring algorithms with low

complexity as they have few decision-points. The further improvement was caused by the great reduction in instructions for processing the control data, owing to the fact that 594 control data were eliminated. It should also be noted that the percentage reduction in complexity of the whole system is higher than the reduction in mean complexity per module. This is due to the fact that the system considers the selection instructions regulating coordination among the various modules [WM96], which has been simplified in the reengineered version thanks to the reduction in the total number of modules.

Figure 9 shows the fluctuations in population of the Residual DB during the course of the reengineering process: empty at the start, it underwent continual filling and emptying due to the combined effect of the Redesign Data and Empty Residual DB phases. Note that at the end of the last iteration the Residual DB is not empty, in that only the 10% of Fa2000 was reengineered: it will be emptied only after the reengineering of the whole system. A record is kept at each iteration of the reengineering process, i_1, i_2, ..., i_{26} of:

- the number of records inserted in the Residual DB during the Redesign Data phase, shown on the graph as a straight line;
- the number of records eliminated from the Residual DB during the Empty Residual Database phase, shown as a dotted line;
- the number of records contained in the Residual DB at the end of the iteration, shown as a dashed line.

It should also be borne in mind that not all the data contained in the Residual DB are removed after each iteration, so the residual population is inherited by the successive iteration.

Finally, the graphs in Fig. 10 compare the number of copy and data files used by a program (Fdittb.cbl) before (Fig. 10.a) and after (Fig. 10.b) reengineering. It can be seen that the large number of files used in the legacy version has been drastically reduced in the reengineered version. This is because management of the files has become more localized, while the pathological files have been eliminated.

6. Conclusions

The paper describes a process of gradual reengineering of the procedures in a legacy system. This process is integrated and completed by the data reengineering process already described by the authors. The process has been successfully experimented on a subset of an industrial legacy system. The obtained results was so encouraging that the customer decided to entrust us with the reengineering of the whole system. Therefore, one of our future works will aim at carrying out the reengineering of the whole legacy system.

The data obtained during experimentation of the system demonstrate that the legacy system was kept working throughout the reengineering process. Moreover, only few requests for change had to be held up and this delay lasted only a few days. Finally, an example of an adaptive change realized during reengineering is shown. The experiment shows the dynamics of the residual data and those transferred to the new database, demonstrating that the two databases, the legacy system and the restored system, could coexist. Some examples showing the coexistence of two operative systems and three systems of functional components, the legacy, the restored and the reengineered systems, are also shown. The quality measurements made before and after the reengineering process show that this process have solved all the aging problems we identified in the legacy system.

Note that in our experimentation we did not take into account the performance loss [Sne99] due to the introduction of the data banker. Besides of this, the main weaknesses of the approach are:

- the need to build and maintain the data locator, which will then be removed at the end of the reengineering process; this weakness can be minimized by reusing the programs included in the data banker;
- the need to manage the residual database, which also has to be removed after completion of the reengineering; however, at least the approach makes this management transparent for the system maintainer.

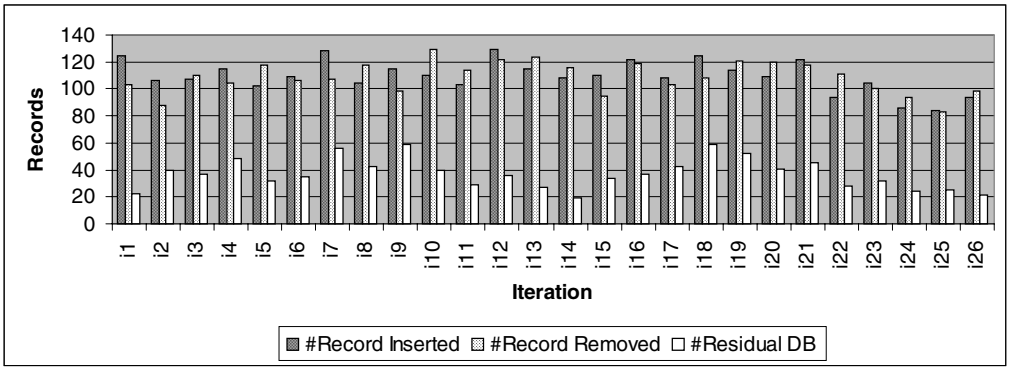
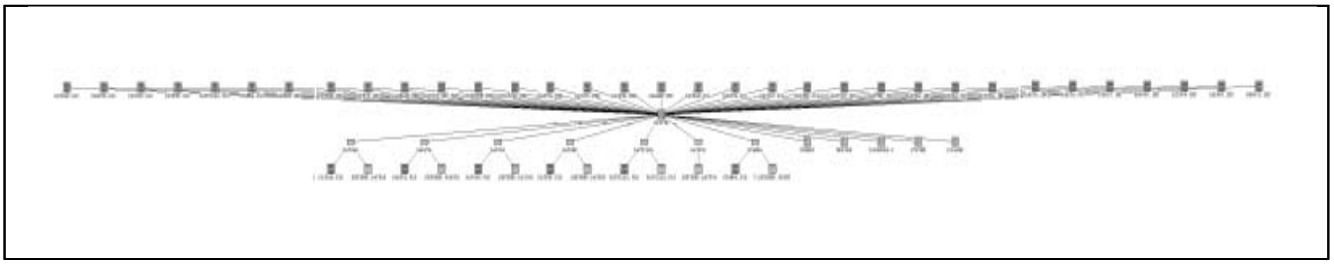
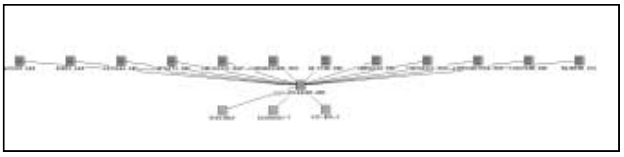


Figure 9. Progress of the Residual DB population during the reengineering process



a



b

Figure 10. Comparison of the copy files and the data files used in the original version (a) and the reengineered version (b) of program Fdittb.cbl

Acknowledgments

We would like to thank the students who participated in the case study, and mainly Dott. G. Lanotte, for their fruitful work. We are also grateful to the anonymous referees for their insightful remarks.

References

[Acu00] AcuCORP, "AcuCOBOL - GT", http://www.acucorp.com/Solutions/acucobol-gt.html, 2000

[BCV00] A. Bianchi, D. Caivano, G. Visaggio, "Method and Process for Iterative Reengineering Data in a Legacy System", *Proc. Working Conf. on Reverse Engineering*, 2000, pp. 86-96.

[Big89] T.J. Biggerstaff, "Design Recovery for Maintenance and Reuse", *IEEE Computer*, July 1989.

[BLW99] J. Bisbal, D. Lawless, B. Wu, J. Grimson "Legacy Information Systems: Issues and Directions", *IEEE Software*, Vol. 16 No. 5, 1999, pp. 103-111.

[Bro93] A.J. Brown, "Specification and Reverse Engineering", *Software Maintenance Research and Practice*, J. Wiley & Sons, Vol.5, 1993, pp 147-153.

[BS95] M. Brodie, M. Stonebraker, *Migrating Legacy Systems: Gateways, Interfaces and the Incremental Approach*, Morgan Kaufman Publishers Inc., San Francisco, 1995.

[CC90] E.J. Chifosky, J.H. Cross II, "Reverse Engineering and Design Recovery: a Taxonomy", *IEEE Software*, 1990.

[Coy00a] F.P. Coyle "Does Cobol Exist?", *IEEE Software*, Vol. 17 No. 2, 2000, pp. 22-36.

[Coy00b] F.P. Coyle "Legacy Integration Changing Perspectives", *IEEE Software*, Vol. 17 No. 2, 2000, pp. 37-41.

[HHL97] E.R. Hughes, R.S. Hyland, S.D. Litvintchouk, et al., "A Methodology for Migration of Legacy Applications to Distributed Object Management", *Proc. of the Int. Enterprise Distributed Object Computing Conf.*, 1997, pp.236- 244.

[Mer00] MERANT "MicroFocus Products - Revolve", http://www.merant.com/products/microfocus/revolve/, 2000.

[NNB98] W.B. Noffsinger, R. Niedbalski, M. Blanks, N. Emmart, "Legacy Object Modeling Speeds Software Integration", *Comm. of the ACM*, vol.41, no.12, 1998, pp.80-89.

[PRS94] J. Preece, Y. Rogers, H. Sharp, et al., *Human-Computer Interaction*, Addison-Wesley, 1994.

[Qui95] A. Quilici, "Reverse Engineering of Legacy Systems: a Path toward Success", *Proc. of the 17th Int. Conf. on Software Engineering*, 1995, pp.333-336.

[Rob97] P. Robertson, "Integrating Legacy Systems with Modern Corporate Applications", *Comm. of the ACM*, vol.40, no.5, 1997, pp. 39-46.

[Smi85] H.C. Smith, "Database design: composing fully normalized tables from a rigorous dependency diagram", *Comm. of the ACM*, vol. 28, no. 8, 1985, pp.826-838.

[Sne95] H. M. Sneed, "Planning the Reengineering of Legacy Systems", *IEEE Software*, 1995, pp24-34.

[Sne96] H.M. Sneed, "Encapsulating Legacy Software for Use in Client/Server System" *Proc. Working Conf. On Reverse Engineering*, 1996, pp.104-119.

[Sne99] H.M. Sneed "Risks Involved in Reengineering Projects", *Proc. Working Conf. On Reverse Engineering*, 1999, pp.204-211.

[Vis97] G. Visaggio, "Comprehending the Knowledge Stored in Aged Legacy Systems to Improve their Qualities with a Renewal Process", *Technical Report ISERN-97-26*, 1997, http://www.iese.fhg.de/network/ISERN/pub/technical_reports/isern-97-26.pdf.

[WLB97] B. Wu, D. Lawless, J. Bisbal, et al., "The Butterfly Methodology: A Gateway-free Approach for Migrating Legacy Information System", *Proc. Int. Conf. Eng. Complex Computer Systems*, pp. 200-205.

[WM96] A.H. Watson, T.J. McCabe, "Structured Testing: A Design Methodology Using the Cyclomatic Complexity Metric", *NIST Special Publication 500-235* – NIST Contract 43NANB517266, D.R. Wallace (Ed.), September 1996.