Software Re-Documentation Process and Tool

Nicolas Anquetil, Kathia M. Oliveira, Anita G.M. dos Santos, Paulo C.S. da Silva jr., Laesse C. de Araujo jr., and Susa D.C.F. Vieira

UCB – Catholic University of Brasília, Brasília, Brazil
{anquetil,kathia}@ucb.br

Abstract. Researchers and professionals know the importance of the documentation for the efficient maintenance of legacy software. Unfortunately, many legacy systems lack this important artifact. Maintenance then becomes a difficult process where software engineers must study and understand the system over and over again. A possible solution out of this situation is to re-document the legacy system. In this article we will present a software re-documentation process, its main features, and constituting activities. We will also present a tool we are developing to automate this process as much as possible. This tools runs in Java and is currently designed for Visual Basic legacy systems.

1 Introduction

It is an accepted fact that legacy software systems are generally poorly documented. This fact makes it extremely difficult to understand and maintain such systems. Redocumenting them could be a great help to keep them "alive". In this paper, we describe a redocumentation process we designed and our first efforts to automate it. A tool, called Redoc, will be described which currently automates two of the activities of our process for legacy systems written in Visual Basic.

In the following sections, we will first present the software redocumentation process (section 2). Then we discuss some existing approaches to redocumentation with a focus on existing tools (section 3). In section 4, we present the Redoc tool. And finally we propose our conclusions and possible future work.

2 The Redocumentation Process

We were called to help redocumentating the main system of an organization. for this, we had to define a redocumentation process. Common sense imposed that the process should have the three following characteristics:

- Reverse engineering process: A redocumentation process should be based on a bottom-up approach, taking advantage of the existing code.
- Light weight documentation: To lower the costs and maximize the chances of the recreated documentation being maintained afterward, we will follow Pressman's recommendation [4, p.807] to limit it to the minimum required.

Proceedings of the CAiSE'05 Forum - O. Belo, J. Eder, J. Falcão e Cunha, O. Pastor (Eds.)

– Good quality/price ratio: We tried to favor documentation artifacts that could be produce automatically or semi-automatically and still offered valuable information for maintenance.

As illustrated in Figure 1, our process is composed of three main phases which include seven activities:

Preparation Phase: analyze the state of the software and its documentation.
Planning Phase: decide what parts of the system should be redocumented first and what will be the general approach.
Redocumentation Phase: recreate the various documents, it constitutes the core of the redocumentation process.

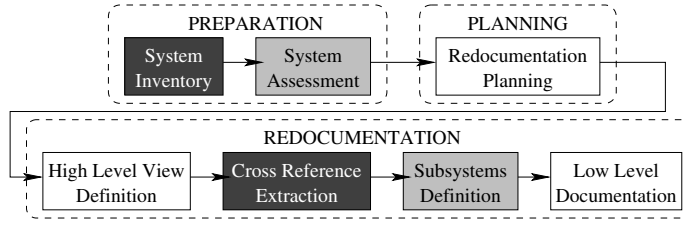

Fig. 1. Activities of the Redocumentation Process (black box: activity automated[1], gray box:activity is partially automated)

To keep the documentation to a minimum, we decided to do mostly without what we call the intermediate documentation which, during development, would be generated during the analysis and design activities.

We try, in the process, to concentrate on what we call the high level and the low level documentation. Traceability between these two levels is guaranteed by a set of cross references (e.g. between implemented funcionalities and implementing routines) extracted automatically.

There are two activities to the Preparation phase:

System Inventory: The goal of the first activity is to get an idea of the size of the problem and provide basic information needed in the following activities. It answers questions like: What exactly constitutes the system? What is known about it? Where to find these informations and new ones? The inventory is performed along three main axes: (i) software components and functionalities, (ii) documentation and (iii) people.
System Assessment: The second step consists in assessing the level of confidence one can have in the code, the documentation and the other sources of information. This is useful to plan the redocumentation in itself and the maintenance in general.

[1] The automation of some activities will be discussed in section 4

There is only one activity to the Planning phase:

Redocumentation Planning: This activity is the prelude to the redocumentation work in itself. It consists in defining how the redocumentation will be performed and what are the priorities. The planning will be based on results of the preceding phase. Other important points to consider are the maintenance load expectancy and the strategic evaluation of the importance of each part of the system.

There are four activities to the Redocumentation phase:

High Level View Definition: In this activity, one must document a first high level view of the system: functionalities, interaction with other systems or specific hardware, etc. Each functionality listed in the System Inventory should be shortly described.

Cross References Extraction: This activity will result in the identification of cross references: "routine to routine" (call graph), "routine to data" (CRUD table), "data to data" (data model), and "functionality to routine". It will be important to do such things as impact analysis, feature location, etc.

Subsystems Definition: This activity will result in a top down view of the system, its subsystems and their components. If the architectural decomposition is known and agreed upon by all, each subsystem listed in the System Inventory must be documented, describing its objective, and what components and funcionalities it contains. In case there is no agreed upon decomposition, we propose to create one using some clustering algorithm (e.g. [1,3,7]).

Low Level Documentation: In this final activity, each independent item identified during the planning will be commented. This activity is to a large extent a manual one, the software engineers must consider each item independently, analyse it and document it.

3 Existing Approaches to Software Re-Documentation

Redocumentation is mainly a problem for large systems where the size alone is already a significant complexity factor. This is one of the reason why there has been a lot of work on automation of this task over the years.

Freeman and Munroe, in [2], discuss some requirements for a redocumentation tool and what documents should be produced during redocumentation.

There already exists some tools to help redocumenting. The simplest would be the tools to extract some documentation from the source code (e.g. javadoc). These tools extract the signature of classes, methods, etc. and sometimes also format comments.

Rajlich [5] proposes a tool to incrementally generate an hypertext documentation of a software as it is maintained. But there is no specific process for redocumentation .

Rigi [8] is a reverse engineering environment to help understand, restructure, and visualize the components of a legacy system. It could help in a redocumentation effort, but it is primarily a program comprehension tool and it does

not, in itself, specify how one goes about redocumenting. A more organizational approach is adopted in [6], where Tilley , specify some requirements for re-documentation and show how Rigi could help in producing it. However the work does not specify a process (sequence of steps).

4 A Software Re-Documentation Environment

We started to develop a software environment to support software engineers in the execution of the various activities of the redocumentation process. The "Redoc" environment has two goals:

- First, it should guide its users through the execution of the various activities, allowing them to register the result of these activities.
- Second, it should provide automate as much as possible the activities of the process that may be automated.

The Redoc environnement is still in an early stage. It is developed in java using the graphical library Swing. It currently parses systems written in Visual Basic and implements the two activities of the software redocumentation process which may be automated (in black in Figure 1).

In the System Inventory activity, one must list the components of the system, and its functionalities. Since Visual Basic (VB) is Object Oriented[2], the components will be classes and their methods. One must also list the tables that make up the system.

The Redoc environment uses javacc[3] to parse the Visual Basic code:

- Extraction of classes and methods is straightforward, they are readily available in the language grammar.
- To discover all the functionalities implemented in the system, we use the menus of the application. This is possible because, in VB, the graphical interface is built through a tool which generates the code in a standardized way. Other languages may raise more difficulties.
- To identify the tables used in the system, we also parse the code to identify the SQL queries it contains and to what tables they refer. Usually the SQL queries are manually programmed and do not follow the same strict patterns as graphical instructions do. They may also be dynamically constructed in the program from data entered by the user. This would make them impossible to be automatically analyzed in the general case. Fortunately, in practice, SQL queries are dynamic only with regard to the values of the columns, and not the tables accessed. This allows our approach to work in most cases.

Figure 2, left part, presents a snapshot of the inventory window. The window shows the functionalities, the classes and their methods, and the tables. All these informations are extracted automatically. The rigth part of the figure presents a snapshot of the window to enter (manually) a new contact person. A similar window exist for documents.

[2] Actually, VB is not truly OO, but it does contain classes, methods ...

[3] https://javacc.dev.java.net/

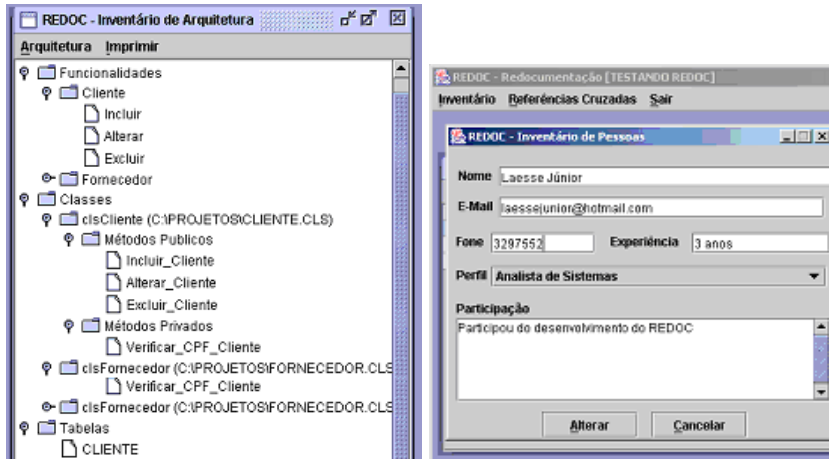

Fig. 2. Result of the automated System Inventory for a system in Visual Basic (left) and a window to enter a new contact person that may help in the redocumentation process (right)

These two examples illustrate the two goals of the environment: automating some activities for the user (left part of the picture), and keeping track of the realization of the activities and registering their result (right part).

The second automated activity is the Cross Reference Extraction. There are four types of cross references:

Data X data: The data X data cross-reference corresponds to finding the relations between the tables used in the system. This is done, again, parsing the SQL queries to detect the joins made between tables.

Routine X Data: The routine X data cross-reference is easy to compute once the table inventory problem is solved. Knowing what tables are accessed from the SQL queries, it is simple to know in what method this query occurs and therefore which methods access which tables. A bit more difficult is to built a CRUD table where each access is marked as Create, Read, Update, or Delete. For this, the tool analyzes the first word of each query (insert, select, update, or delete).

Routine X routine: The routine X routine cross-reference is a simple call graph among the methods and offers no special difficulties.

Funcionalidade X routine: The Functionalities X routine cross-reference consists in identifying what routines implements a functionality. As we identify functionalities from the application menus, it is a simple matter to identify the starting point of a functionality and then compute transitive closure on the call graph from that point. However, there is more to it than that, because a functionality will usually call one or several windows where the actual execution of the functionality will be triggered by clicking a button.

5 Conclusion and Future Work

It is generally accepted in software engineering that most of legacy software suffer from a lack of up-to-date documentation. Redocumentation is the natural solution to help maintaining these software systems. However, there is little work on how this can be done and what tools we need to actually redocument.

In this article, we presented a process for software redocumentation and the steps we are taking to automate it as much as possible. We are developing a software redocumentation environment that will (a) help people register the results of the various activities of the process, and (b) help people gathering the information they need by automating some activities.

Two activities (System Inventory and Cross-Reference Extraction) have already been automated and we are now working on a new project to help automate a third activity (System Quality Assessment)

Acknowledgment

This work is part of the "Knowledge Management in Software Engineering" project, which is supported by the CNPq, an institution of the Brazilian government for scientific and technological development.

References

1. Nicolas Anquetil and Timothy C. Lethbridge. Experiments with Clustering as a Software Remodularization Method. In *Working Conference on Reverse Engineering*, pages 235–255. IEEE, IEEE Comp. Soc. Press, Oct. 1999.
2. Robert M. Freeman and Malcolm Munro. Redocumentation for the maintenance of software. In *Proceedings of the ACM 30th Annual Southeast Conference*, pages 413–16. ACM, ACM Press, Apr 1992.
3. Arun Lakhotia. A unified framework for expressing software subsystem classification techniques. *J. of Systems and Software*, 36:211–231, Mar 1997.
4. Roger S. Pressman. *Software Engineering: A Practitioner's Approach*. McGraw-Hill, 5th edition, 2001.
5. Václav Rajlich. Incremental redocumentation using the web. *IEEE Software*, 17(5):102–6, Sep 2000.
6. Scott R. Tilley. Documenting-in-the-large vs. documenting-in-the-small. In *Proceedings of CASCON'93*, pages 1083–90. IBM Centre for Advanced Studies, Oct. 1993.
7. Theo A. Wiggerts. Using Clustering Algorithms in Legacy Systems Remodularization. In *Working Conference on Reverse Engineering*, pages 33–43. IEEE, IEEE Comp. Soc. Press, Oct. 1997.
8. Kenny Wong, Scott R. Tilley, Hausi A. Müller, and Margaret-Anne D. Storey. Structural redocumentation: A case study. *IEEE Software*, 12(1):46–54, Jan 1995.