www.ijecs.in
International Journal Of Engineering And Computer Science
Volume 8 Issue 5 May 2019, Page No. 24621-24628
ISSN: 2319-7242 DOI: 10.18535/ijecs/v8i05.4317

Extracting Business Rule from existing COBOL programs for Redevelopment

Ekwonwune Emmanuel N[1] and Egwuonwu Deborah I[2]

[1]Department of Computer Science, Imo State University, Owerri Nigeria
[2]Department of Computer Science, Abia State Polytechnic, Aba, Nigeria

Abstract
The paper describes an industrial business study carried out to regain the business knowledge embedded in a legacy COBOL application. The aim of this work was to extract out the information required to re-implement the Legacy programs in a new client/server environment. The progress solution is in four step. Firstly, the programs were restructured, secondly the programs were sliced into business logic modules, third the business modules were subjected to a multi view analysis and finally the views were integrated into a unified documentation describing the data, decision and procedural flow of each program slice.

Keywords: Reengineering, Reverse Engineering, Knowledge Extraction, Business Rules, Program Comprehension, Slicing

1. Introduction

A legacy system is one which is extremely valuable to an organization, performing key strategic functions. But maintaining such systems as to incorporate new functionalities or due to organizational policy changes become hard in the absence of proper documentation, qualified staff and other resources. Due to the poor quality of the code, it is better to redevelop the application using modern object- oriented techniques and this called for creating a new functional specification and a new architectural design as it remain many black holes. Therefore if a description of the detailed business logic is to be recovered, then it must be recovered from the programs as the only reliable source of information. The cost of re-engineering a system is generally less than developing a new system. Sometimes what is required is, add some functionalities, change some policies, change the structure of system without changing functionalities, changing the architecture of the system to add some non functional requirements and for making these changes, it is worthless to develop a new system.

Warren.[1999]. Stated that a legacy system is "an old system which remains in operation within an organization" In the same year, Alderson *et al.* [1999] discussed two more definitions for legacy systems: "any code that has left development" and "a system whose security has been compromised". For many business-critical systems the problem was how to renovate the software system while at the same time business continued as usual. An often heard solution was to throw away the software as soon as a totally new system was finished; this 'was sometimes called shadowing (Van den Brand *et al*., 2007).

2. Literature Review

Chikofsky and Cross II [1990] stated that reverse engineering is "the process of analysing a subject system to identify the system's components and their interrelationships and create representations of the system in another form or a higher level of abstractions"

Osuagwu, *et al*., (2008) had defined RE as the process of analyzing a subject system to create representations of the system at a higher level of abstraction in order to unravel the complexities of target software or to unveil the assumptions made by the people who created the system and then undermine those assumptions

Perry *et al*, (2004) in their book Introduction to Reverse Engineering Software defined RE as "simply the act of figuring out what software that you have no source code for does in a particular feature
Chung and Le (2000) defined RE as a process to transform a code into model through a mapping from a specific implementation language.
Reverse engineering reduces the risk and cost of software development, Aiken.et al (1993).
Reverse engineering is the process of analyzing a subject system; first to identify the system's components and their interrelationships and second to create representations of the system in another form or at a higher level of abstraction [Jain. 2011].
According to [Tilley. 1994], a reverse engineering approach should consist of following steps: Extraction: extract information from source code, documentations. Abstraction: abstract the extracted information. and Presentation: transform abstract data into a representation.
Re-engineering frameworks like [Harandi and Ning. 1990; Wilde and Huitt 1992] indicate how object-oriented development methods (as a representative of modern software development) can be used to re-engineer procedural legacy code. Much effort has been invested in the re-engineering of COBOL code to transform programs written in COBOL-74 to COBOL-85 [Sneed and Jandrasics. 1987].

3. Overview of Legacy Software

This work is intended to develop a Re- engineering method to automate the extraction of business rules from Source code. Extracted business rules can be classified as structural, behavioral and constraint rules. In this, a program has been taken as input and then candidate variables are identified. The available tools left the task of identification of variables on the maintainer. And good approximation of variable is necessary of extraction of required business rule. The identified variable will be used for program slicing. The output of program slicing will be sliced segments which will contain the business rule. These sliced segments will be than given to presentation tool to present the rule in different views so that different stakeholder can easily understand the rules.

COBOL is unstructured program and nature event driven. The technique for connecting part of the COBOL code is the GO TO branch. Loops are implemented and identified using backward GO TO. The IF statements, are terminated by a period, and structured statements such as EVALUATE and IF...END-IF indicate recent patches. COBOL programs has no information hiding or abstract data type. Each program has an extensive global area with thousands of variables. The procedure divisions are segmented into sections and GO TO's branch outside of a section, but instead are collected at the end mode. This makes it possible to separate sections from control logic. Therefore, the programs uses procedural slicing. They can be split up into a main section and several subsection.

4. Re-engineering

Re-engineering helps in better understanding of system even if documentation is not available. Re-engineering a system is generally less expensive than developing a new system . A re-engineering method can help in extraction of business rules buried in source code of legacy system. The extracted business rule must be at high level of abstraction so that every stakeholder of the system could understand the system. an organization provides documents that contain business rules for development of a system. Various changes have been done on the system to maintain the system. This results in disparity among the document and the software, as proper documentation was not done along with the maintenance work.

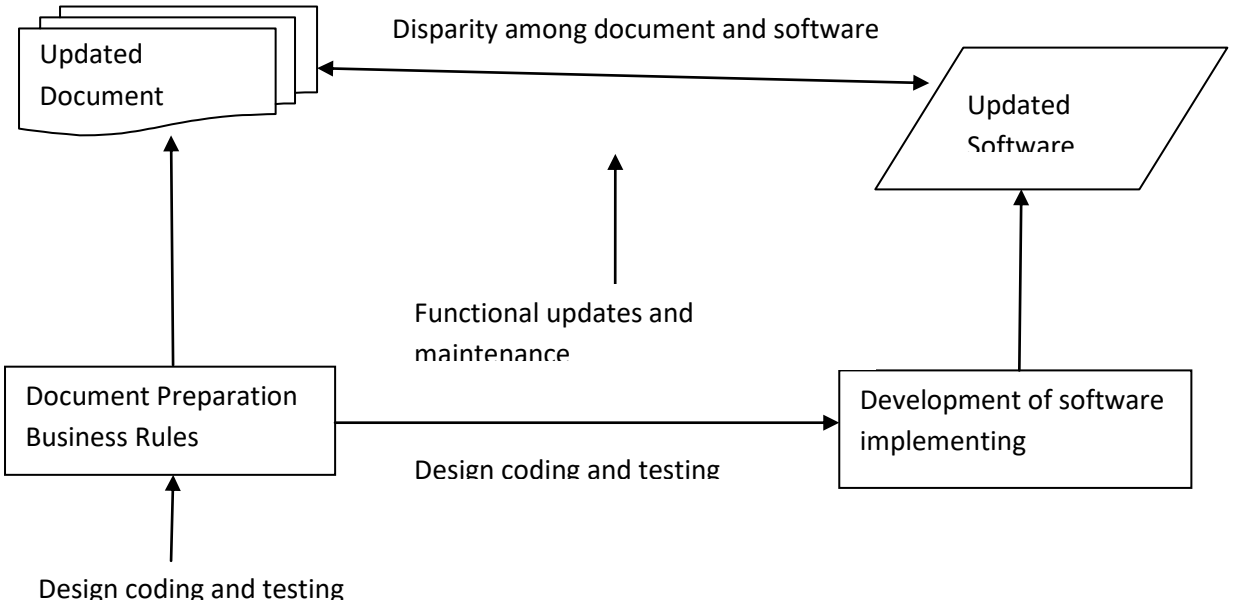

Fig.1: Re-engineering

5. Extraction Method

The major portion of the project was devoted to the development of a set of tools to support the automated knowledge acquisition process, a process consisting of four sequential steps (Hanna, 1974). 1. To restructure the procedural code to facilitate slicing.

2. To slice the code into partial programs each processing a discrete business rule.
3. To generate a set of views on each partial program.
4. To integrate these disjointed views into a single unified business rule documentation. See diagram

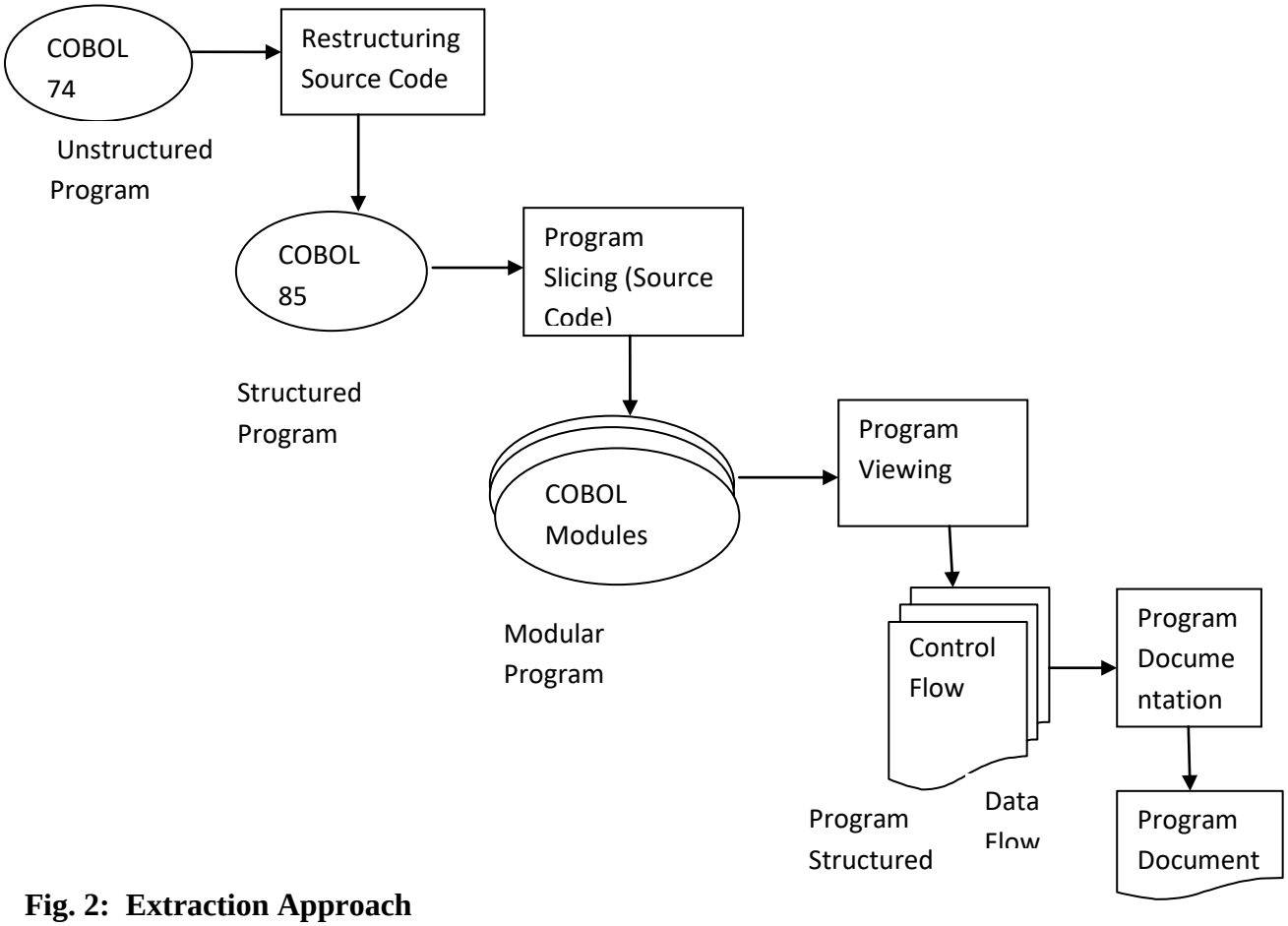

Fig. 2: Extraction Approach

Restructuring Source Code

Source code restructuring was fully automated, according to (Sneed 1998), It uses SoftRecon tool to :reformats the procedural code splitting the lines with more instructions and indenting the nested code,

- it cuts the input/output operations and database accesses out of the mainline code and pastes them on the end of the program in a separate data access section,
- it removes obsolete and dangerous statement types such as the PERFORM THRU, ALTER and NEXT SENTENCE replacing them with standard statements,
- removes the periods at the end of the IF statements, replacing them with END-IFs for each IF..ELSE pair,
- it removes all of the GO TO branches, replacing them with a label variable assignment to PERFORM the paragraph to which the GO TO is branching without returning,
- it recognizes backward branches and converts them to a PERFORM UNTIL loop construct.

According to Hay at el (2000); Extraction of business rule is not a fully automated process. Understanding a legacy system requires automated and manual analysis.

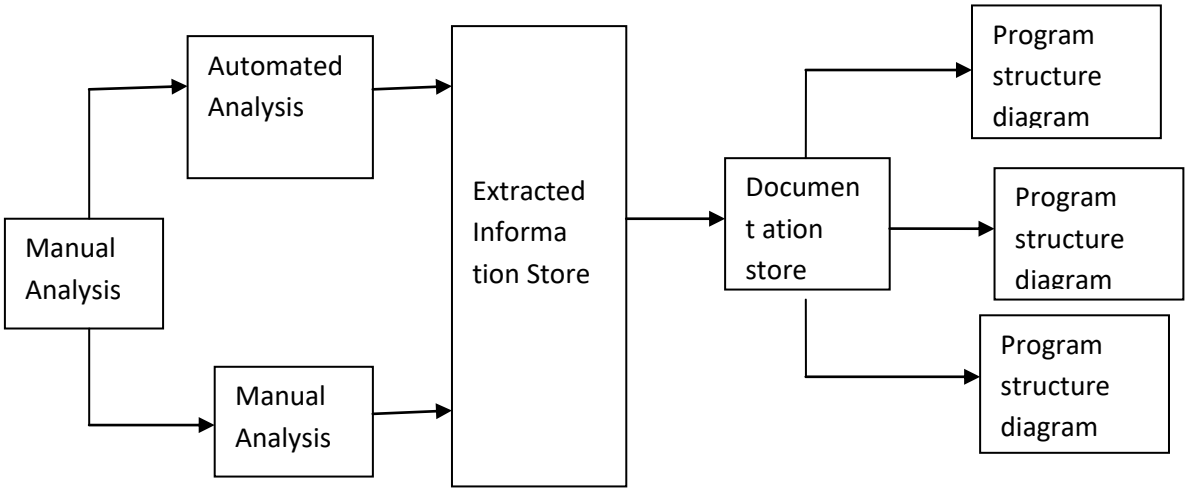

Fig.3: Analysis of legacy system

The system provides the user with flowchart of Individual Programs to better understand the flow of process steps. Extraction of Business Rules using is done using "Variable" filters. each variable is assignment, an appropriate descriptions for better understanding and usage. The output as Business Rules are presented in Natural Language (ENGLISH). For instance.

Use Case driven Program Slicing

In Program Slicing human intelligence was used to identify the logical entry points, that is those points where the processing of a particular use case begins. It may be a function, a label or a procedure entry. A use case is related to the main section with several sub sections. The user of the tool needs to mark the source line where the slice begins. It may be the point where an input panel is received or it could be the beginning of a processing loop. The rest is taken care of automatically by means of a recursive invocation algorithm in the tool COBWrap. Each PERFORM from the original code slice is pursued to include that path in the slice. If the included slice contains additional references to other code slices these too will be pursued and the affected code included until all PERFORMs have been resolved.

After the said source segment has been cut out of the procedure division, a data flow analysis is performed to recognize all data variables processed by that segment. These variables are then marked in the Data

Division together with the structures they are included in to create a new reduced Data Division for the sliced code containing only those variables used by that code. The same process is repeated for the files in the Environment Division. The end result is a partial program consisting of all procedures and data required to process a given use case. The reduced data structure is placed in the Linkage Section to be passed as a parameter from the calling program. This second step is repeated for each unique use case, so that there are several partial programs created from the same original program. Code traversed by many rules is duplicated and included in each partial program. The procedural slicing technique was introduced by (Weiser 1984) and extended by(Cimitile 1995).

6. Methodology

Business rule represents structure, behaviour or constraints of an organization. So to implement a business rule through a program what is required is some input, conditions, assignments, results. Formally a business rule is a code segment in program where a input is checked based on some condition using conditional operators and result either assigned after calculation or in form of Boolean yes or no to output variable e.g. Taking a COBOL example that is well known :

```
IF HRS-WORKED > 42
    COMPUTE  TOTAL-SALARY = 42 * WAGE-RATE
        + ( HRS-WORKED - 42 ) * 2 * WAGE-RATE
ELSE        COMPUTE TOTAL-SALARY = HRS-WORKED * WAGE-RATE.
```

To extract a business rule we identify the candidate variables, and then use program slicing to extract the code segment.

Program slicing to extract the code

- To identify the data (out put variable) that implements the business rule.
- To find the reference (Assignment reference) of the data in which it is used in the program.

Assignment references are the place where data are created or altered, so they are the key place for extracting the business rules.

New_price = old_price + (old_price x sale_tax);

So here a business rule is implemented that new price will be old price of the commodity plus sale tax over the price of the commodity. But collections of these assignment statements is not a simple task as these assignment statements are scattered along the program. E.g.

```
1. else if(a== la && f < lf) {
2. if(ff[f] == ' ' )
3. ans[lans] = '_';
4. else
5. ans[lans] = 'x';
6. convert(a, f+1, abb, ff, la, lf, ans, lans+1 );
 7. return;
8. }
9. else if (a < la && f == lf )
10. return;
 11. if(abb[a] == ff[f])
12. {
 13. ans[lans]='M';
 14. convert(a+1,  f+1, abb, ff, la, lf, ans, lans+1); 15. ans[lans]='x';
16. convert(a,  f+1, abb, ff, la, lf, ans, lans+1); 17. return;
18. }
19. if(ff[f] == ' ' )
 20. ans[lans] = '_';
21. else 22. ans[lans] = 'x';
 23. convert(a,f+1,abb,ff,la,lf,ans,lans+1);
24. }
25. else if(a== la && f < lf) { 26. if(ff[f] == ' ' )
27. ans[lans] = '_';
28. else
```

In above example if we concentrate on ans [] assignment reference to this variable is made at various lines like

3. ans[lans] = '_';

ans[lans] = 'x';

13 ans[lans]='M';

```
15   ans[lans]='x';

20   ans[lans] = '_';

22   ans[lans] = 'x';
```

So method used to extract business rule implemented with ans [ ] is we will find out the point where these assignment references are triggered. We have maintained a tree structure were we maintain a relation of assignments and triggering conditions. The assignment statements are captured using static forward program slicing. E.g

```
11. if(abb[a] == ff[f])

12.{

 13  ans[lans]='M';

 15 ans[lans]='x';

18 }

 if(ff[f] == ' ' )

 ans[lans] = '_';

else ans[lans] = 'x';
```

These are the slices we get now the main task now is to represent these extracted program segments in a form that may be easily understood by a reader.

References

[1] Ian Warren (1999), The Renaissance of Legacy Systems. Springer London,
[2] Alderson and H. Shah (1999), "Viewpoints on Legacy Systems," vol. 42, no. 3, pp. 115–116,
[3] M. van den Brand, P. Klint, and C. Verhoef, "Re-engineering Needs Generic Programming
[4] Language Technology," SIGPLAN Not., vol. 32, no. 2, pp. 54–61, 1997. [Online]. http://doi.acm.org/10.1145/251621.251633 17
[5] E. J. Chikofsky and J. H. Cross, "Reverse engineering and design recovery: a taxonomy," IEEE Software, vol. 7, no. 1, pp. 13–17, Jan 1990. 4, 5
[6] Osuagu, O, E, Oladipo,O. F and Banjo C. (2008). Deploying Reverse Software Engineering as tool for converting Legacy Application I critical –sensitive systems for Nigerian Industries. In

Proceedings of the 22nd National conference and AGM of the Nigeria Computer Sociaty Conference (ENCTDEV 2008), 24-27 june.
[7] Perry, M, Nasko O. (2004). Introduction to Reverse Engineering Software . Downloaded from http://www.acm.uiuc.edu/sigmil/RevEng/ch01/html
[8] Chung A, Lee Y.S,(2000) "Reverse Software Engineering with UML for website maintenance," IEEE proceedings – Working Conference in Reverse Engineering, '00, pp.157-172,2000.
[9] Jain, S. (2011), "Reverse engineering: Journey from code to design," Electronics Computer Technology (ICECT), 3rd International Conference on , vol.5, no., pp.102-106, 8-10
[10] S. Tilley, K. Wong, M. Storey, H. Muller, Programmable reverse engineering, International Journal of Software Engineering and Knowledge Engineering 4 (4) (1994) 501–520.
[11] Wilde N.& Huilt R (1992) Maintenance support for object oriented programs. IEEE Transations on software Engineering, 18(12):1038-1044,, December 1992.
[12] Sneed, H./Jandrasics,G.: "Inverse Transformation of Software from Code to Specifications", Proc. of ICSM-88, IEEE Computer Society Press, Phoenix, Oct., 1998, p. 102
[13] Hanna, A. "Getting back to requirements proving to be a difficult Task", IEEE Software Magazine, Oct. 1994, p. 49
[14] Weiser, M.: "Program Slicing", IEEE Trans. on S.E., Vol. 10, No. 4, July, 1984, p. 352 [9]
[15] Cimitile, A./ de Lucia, A./ Munro, M.: "Identifying Reusable Functions using Specification Driven
[16] Hay D and Anderson Healy K: 'Defining business rules — what are they really?', White paper, The Business Rules Group, Version 1.3 (2000)
[17] Weiser M: 'program slicing', IEEE transactions on software engineering,10,no4,pp 352-357 (1984).
[18] Aiken.P, Muntz, A, Richards, R, "A framework for reverse engineering DoD information legacy systems", Proceedings of Working Conference on Reverse Engineering, 1993.