Advanced Science and Technology Letters
Vol.29 (ASEA 2013), pp.258-263
http://dx.doi.org/10.14257/astl.2013.29.54

Flowchart Knowledge Extraction on RPG Legacy Code

Kochaporn Suntiparakoo and Yachai Limpiyakorn,

Department of Computer Engineering, Chulalongkorn University, Bangkok 10330, Thailand
Kochaporn.Su@student.chula.ac.th, Yachai.L@chula.ac.th

Abstract. RPG was originated as a report-building program developed by IBM. Many business applications are written in RPG, and they are often critical in the operations of enterprises. Applications written in RPG can be considered as legacy software. Through decades of use, these RPG legacy systems can be hard to maintain, improve, and expand, since there is a general lack of understanding of the systems. The supporting documentation may not be current as well due to many changes implemented into the software. This paper thus presents a method of flowchart knowledge extraction on RPG legacy code. The metadata is gathered from the input text file, then processed and mapped to DOT markup language format for flowchart rendering using visualization tool named Graphviz. The prototype implemented in this work would facilitate the understanding of RPG legacy code during software maintenance process.

Keywords: software maintenance, legacy system, RPG language, metadata.

1 Introduction

Legacy software can be characterized as old software that is still performing a useful job. Through years of use, users are familiar with the look and feel of the system, and are reluctant to change. These archaic codes have been developed and maintained for many years, and they have become part of the integral business environment. The new replacement system may not fulfill the business requirements, and the investment may be prohibitive.

RPG (Report Program Generator) [1] is the programming language developed by IBM in 1959. RPG was originated as a report-building program used in DEC and IBM minicomputer operating systems, and evolved into a fully procedural programming language. Software developed with RPG language (except RPG IV) can be considered as legacy software. Nevertheless, many business applications are written in RPG, and they are often critical in the operations of enterprises, namely software used in commercial bank and production line control. For decades of use, these RPG legacy systems can be hard to maintain, improve, and expand, since there is a general lack of understanding of the system. The developers who were experts on it have retired or forgotten what they knew about it. This can be worsened by loss or lack of updated documentation. The study reported that organizations have spent 20% to 70% of computing effort on maintenance tasks [2].

ISSN: 2287-1233 ASTL

This paper thus presents a method for knowledge extraction from RPG legacy code. The intent of code will then be visualized as flowchart, which is a schematic representation that illustrates a sequence of operations. Flowchart can be used as the program specification document to serve software maintenance activities.

2 RPG (Report Program Generator) [1]

RPG is a structured programming language. Programmers must be concerned about the position of code when writing RPG statements. RPG/400 is composed of seven specifications, each of which must be outlined in the following sequence:

1. Control Specification (H). provides information about the program.
2. File Description Specification (F) defines all files in the program.
3. Extension Specification (E) describes arrays, tables.
4. Line Counter Specification (L) indicates the length of overflow lines.
5. Input specification (I) describes data structures, named constants, records, and fields in the input files; and indicates how the records and fields are used by the program.
6. Calculation Specification (C) describes the program computations and indicates the order in which they are done. Calculation Specifications can control certain input and output operations.
7. Output Specification (O) describes the records and fields, and indicates when they are to be written by the program.

An RPG program typically starts with File Specification, listing all files being written to, read from or updated; followed by Extension Specification containing program elements such as data structures and dimensional arrays; then followed by Calculation Specification, which is the computation part including record matching to generate reports from data files. Finally, Output Specifications can follow to determine the layout of other files or reports. Example RPG/400 source code, CSTOCK, is shown in Fig. 1.

3 Research Methodology

3.1 Preprocess

This step aims to create the metadata of the input text file of RPG code. The metadata is data about data to facilitate the discovery of relevant information [3]. It is in the format of machine understandable referring to information contained in the source code that will be analyzed and mapped to visualize the intent of code with flowcharts. This step consists of 2 sub-processes: 1) chunking, and 2) detection of operation code and controls

```
F* ---------------------File definition --------------------- *
FPRODB IF E             K       DISK
FSTOCK UF E             K       DISK
C*Calling parameter definition
C           *ENTRY   PLIST
C                    PARM           #ID     8        NUMBER1
C                    PARM           #ERR 10 0-OK,0<0-NG
C*Key definition
C           #KEY1    KLIST
C                    KFLD           PMODEL
C                    KFLD           PCOLOR
C                    Z-ADD0         #ERR              CLEAR ERR FLAG
C*Check ID in Production order bank file
C           ID       CHAINPRODB                91
C           *IN91    IFEQ *OFF
C           #KEY1    CHAINSTOCK                92
C*Check volume in stock file
C           *IN92    IFEQ *OFF
C           SVOLUM ANDGT0
C                    SUB 1          SVOLUM
C*Update volume of stock file
C                    UPDATESTOCK
C                    ELSE
C                    Z-ADD1         #ERR
C                    ENDIF
C                    ELSE
C                    Z-ADD1         #ERR
C                    ENDIF
C                    SETON                     LR
C*End program
```

Fig. 1. Example RPG source, CSTOCK, as input for flowchart knowledge extraction**.**

3.2 Generate Program Information

The function of this step is to generate program information from File Description Specification and Calculation Specification. Program information contains program name, input/ output parameters, and working files. The input/ output parameters are from Calculation Specification. Output parameters are those updated fields, otherwise, they are considered input parameters. The working files include input files, output files, updated files, workstation files, and report files contained in File Description Specification. The program information of input file CSTOCK (Fig. 1) is illustrated in Fig. 2.


```
Program name is CSTOCK
The input parameter is #ID (8)
The output parameter is #ERR (10)
This program have 2 file declare.
The input file is PRODB
The updated file is STOCK
```

Fig. 2. Program information of input file CSTOCK.

3.3 Generate Detailed Flowchart

The function of this step is to create the markup language file of the detailed flowchart that explains the program logic. The output markup language file is generated by mapping the metadata of RPG source with markup language tags. The metadata is stored in the directed graph as the result from the preprocessing step. To map the metadata with markup language, of which the format is shown in Fig. 3, the details of each node and edge in the directed graph will be accessed to identify the type, symbol, and label of components in the flowchart. The output markup language file will then be fetched into the visualization tool, Graphviz [5]. The markup language used in Graphviz is DOT format with file extended .dot.

```
Node1.ID->Node2.ID[label = "Edge.label"];
Node1.ID [label = "Node.label" ,shape = Node.type.dot]
```

Fig. 3. Format of DOT markup language used to map with metadata.

3.4 Generate Intent of Code

This step generates the high-level abstraction of flowchart that explains the intent of code. To create intent of code, comments will be extracted from the source code. It is observed that intent of code is subject to the quality of comments. A group of RPG instructions will be replaced by a comment to express the intent of a code chunk.

4 A prototype

To support the automation of Flowchart Knowledge Extraction on RPG Legacy Code, a prototype has been developed using Eclipse Kepler 4.3 [6]. The prototype facilitates the detection of operation code and controls in the input RPG source code, the transformation of RPG source into metadata stored in the directed graph, and the mapping of metadata with DOT markup language.

```
Digraph G{
        subgraph custer_Info{
        label = "Program Infomation";
        17[ label = "Program name is CSTOCK \lThe input parameter is #ID(8)
        \lThe output parameter is #ERR(10) \lThis program have 2 file declare.
        \lThe input file is PRODB. \lThe update file is STOCK \l", shape = rect];
        }
        subgraph cluster_Concept{
        label = "Intent of Code";
        18 -> 19[label = "  " ];
        19 -> 20[label = "  " ];
        20 -> 21[label = "  " ];
        21 -> 22[label = "  " ];
        22 -> 23[label = "  " ];
        23 -> 24[label = "  " ];
        18[ label = "Start" ,shape = Start ];
        19[ label = "Calling parameter definition" ,shape = Start ];
        20[ label = "Key definition" ,shape = rect ];
        21[ label = "Check ID in Production order bank file" ,shape = rect ];
        22[ label = "Check volume in stock file" ,shape = rect ];
        23[ label = "Update volume of stock file" ,shape = rect ];
        24[ label = "End" ,shape = process ];
        }
        subgraph cluster_Detailed {
        label = "Detailed Flowchart";
        1 -> 2[label = "  " ];
        2 -> 3[label = "  " ];
        3 -> 4[label = "  " ];
        4 -> 5[label = "  " ];
        5 -> 6[label = "  " ];
        6 -> 7[label = " Yes " ];
        6 -> 14[label = " No " ];
        7 -> 8[label = "  " ];
        8 -> 9[label = " Yes " ];
        8 -> 11[label = " NO " ];
        9 -> 12[label = "  " ];
        11 -> 12[label = "  " ];
        12 -> 15[label = "  " ];
        14 -> 15[label = "  " ];
        15 -> 16[label = "  " ];
        1[ label = "START" ,shape = Start ];
        2[ label = "Receive parameter  \n #ID     8 \n #ERR    1" ,shape = rect ];
        3[ label = "Define Key list of #KEY1\n PMODEL\n PCOLOR" ,shape = rect ];
        4[ label = "Set variable #ERR = 0" ,shape = rect ];
        5[ label = "Search file PRODB by key #ID" ,shape = rect ];
        6[ label = "IF (search file PRODB by key #ID found)" ,shape = diamond ];
        7[ label = "Search file STOCK By key #KEY1" ,shape = rect ];
        8[ label = "IF (Search file STOCK by #KEY1 found)\n AND (SVOLUM.STOCK > 0)" ,shape = diamond ];
        9[ label = "Set SVOLUM.STOCK = SVOLUM.STOCK - 1" ,shape = rect ];
        11[ label = "Set variable #ERR = 1" ,shape = rect ];
        12[ label = "ENDIF" ,shape = rect ];
        14[ label = "Set variable #ERR = 1" ,shape = rect ];
        15[ label = "ENDIF" ,shape = rect ];
        16[ label = "ENDPROGRAM" ,shape = process ];
        }
}
```

Fig. 4. Example DOT markup language file for rendering detailed flowchart.

5 Conclusion

The costs of redesigning or replacing the legacy systems may be prohibitive. Influenced by economic reasons, organizations thus usually opt to keep their outdated systems rather than to modernize them. Users may also prefer an evolutionary rather than a revolutionary approach to modernizing their software. While many changes have been made to the software through years of use, the supporting documentation may not be current. This paper thus presents an approach to automating the construction of flowcharts as design blueprints from legacy source written in RPG/400. The recovery of the intent of code starts with input preprocessing consisting of chunking the source code based on types of specifications, and detecting operation code and controls in Calculation Specification. The details of operation code and controls will then be transformed to metadata stored in the directed graph. Next, the metadata will

be mapped with DOT markup language to render flowcharts with the visualization program, Graphviz.

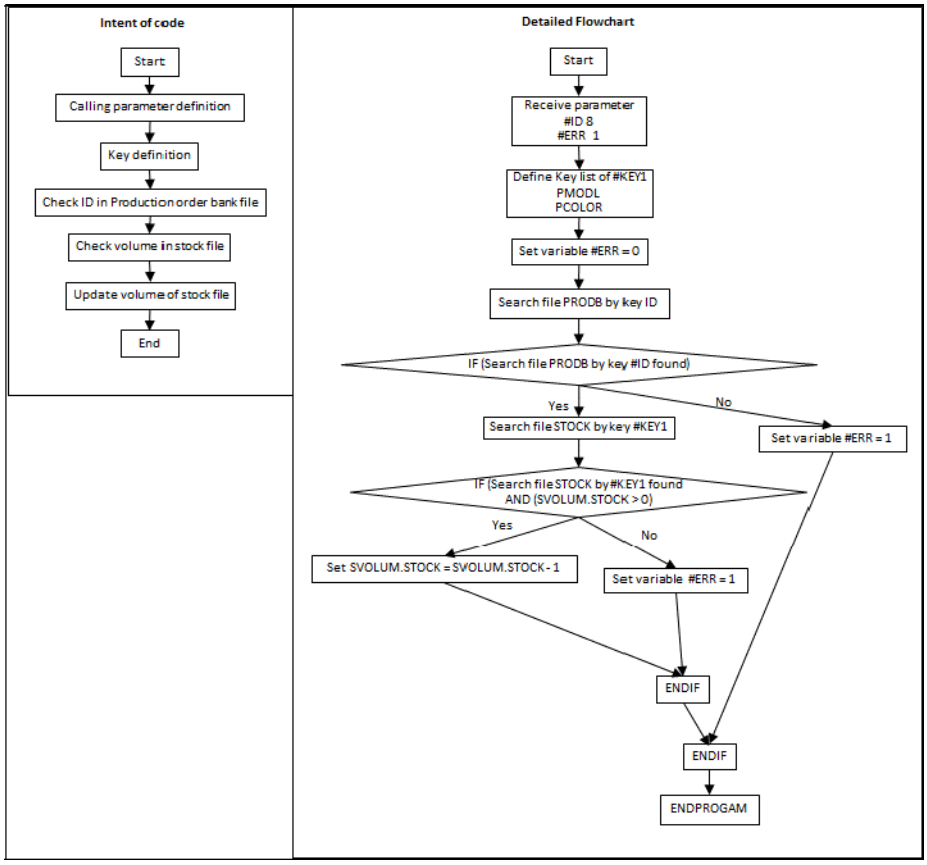

Fig. 5. Flowcharts rendered from Graphviz visualization tool.

References

1. International Business Machines Corporation, http://www.ibm.com/us/en/
2. Lientz, B.P., Swanson, E.B..: Software Maintenance Management. Addison-Wesley, Boston (1980)
3. National Information Standards Organization.: Understanding Metadata. NISO Press, Bethesda (2001)
4. Vasudevan, B.G., Dhanapanichkul, S., Balakrishnan, R.: Flowchart knowledge extraction on image processing. In: IEEE World Congress on Computational Intelligence, pp. 4075—4082. Hong Kong (2008)
5. Graph Visualization Software Document, http://www.graphviz.org/Documentation.php
6. Eclipse Kepler 4.3, http://www.eclipse.org/kepler/