Materials Science Forum Vol. 721 (2012) pp 307-312
Online available since 2012/Jun/04 at www.scientific.net
© (2012) Trans Tech Publications, Switzerland
doi:10.4028/www.scientific.net/MSF.721.307

Reverse Engineering Legacy Finite Element Code

Samuel Ratnajeevan Herbert Hoole[1,a], Thiruchelvam Arudchelvam[2,3], and Janaka Wijayakulasooriya[3]

[1.] Dept. of Elect. and Computer Engineering, Michigan State University, East Lansing, MI, USA
[2.] Dept. of Computing and Information Systems, Wayamba University of Sri Lanka, Sri Lanka
[3.] Dept. of Electrical and Electronic Engineering, University of Peradeniya, Sri Lanka

[a]Correspondence: SRHHoole@gmail.com

Keywords: Reverse Engineering, Reengineering, UML, Class and Sequence Diagrams, Java, FORTRAN, Legacy Software, Finite Elements.

Abstract. The development of code for finite elements-based field computation has been going on at a pace since the 1970s, yielding code that was not put through the software lifecycle – where code is developed through a sequential process of requirements elicitation from the user/client to design, analysis, implementation and testing and release and maintenance. As a result, today we have legacy code running into millions of lines, implemented without planning and not using proper state-of-the-art software design tools. It is necessary to redo this code to exploit new object oriented facilities and make corrections or run on the web with Java. Object oriented code's principal advantage is reusability. Recent advances in software make such reverse engineering/re-engineering of this code into object oriented form possible. The purpose of this paper is to show how existing finite element code can be reverse/re-engineered to improve it. Taking sections of working finite element code, especially matrix computation for equation solution as examples, we put it through reverse engineering to arrive at the effective UML design by which development was done and then translate it to Java. This then is the starting point for analyzing the design and improving it without having to throw away any of the old code. Using auto-translators and then visually rewriting parts by the design so revealed, has no match in terms of speed and efficiency of re-engineering legacy code.

Finite Element Code

Software Engineering, has today matured as a discipline [1-7]. In developing code, strict rules are specified as to how. In a multistage, sequential effort, we start with requirements elicitation from the user/client to design, analysis, implementation and testing (with loops going back from the second stage onwards as dissatisfactions are identified) and finally release and maintenance.

Today as work moves into components-based software with user choice in putting together different methods to do the job at hand [8], it is extremely important that components match and have the correct interfaces. The development of code for finite elements-based field computation has been, correctly speaking, *ad hoc*. Much of the code was in FORTRAN [9] and this code reaching the magnitude of millions of lines could not be practicably redeveloped in more modern languages with their object oriented features.

The state-of-the-art today is code implementing mathematically well-researched and powerful methods developed in research labs to solve particular problems and not the most suitable code in terms of well-designed modules passing the rules of software engineering to have sound interfaces and code design when subjected to standard methods of analysis.

This paper examines the issues of reengineering or reverse engineering this legacy code to bring it into line with the norms of modern-day software engineering principles. Efforts from the late 1990s focused on inspecting finite element code in FORTRAN-77 to inspect the common block to write code with object oriented features [10]. Other efforts in reverse engineering are confined to extracting model features revealed in the code [11] rather than using the code as code.

Re-engineering ancient code to make the code usable, the focus of this paper, becomes necessary if we wish to correct or improve such code, verify the design of the code or use object oriented languages to adopt best practice and facilitate reuse. Indeed, even when the code is developed in C or C++, it may be desirable to re-engineer the code to run in Java so that it may be offered over the Internet [10].

```
public abstract class Matrix
{
    protectedint columns;              // *****  changed
    protectedint rows;                 // *****  changed

publicintrowSize()
{ return rows;}
    publicintcolumnSize()
{ return columns; }

    public abstract void display();
    public abstract double elementAt(int i, int j);
    public abstract Matrix multiply(double K);
    public abstract Matrix multiply(Matrix B);
public abstract columnVector multiply(columnVector
B);
    public abstract Matrix add(Matrix B);
    public abstract Matrix sub(Matrix B);
    public abstract void swapRows(int i, int j);
    public abstract void swapColumns(int i, int j);
public abstract Matrix transpose();
    public abstract Matrix inverse();
}
```

Fig. 1.Generated Java code after modifying the class Matrix

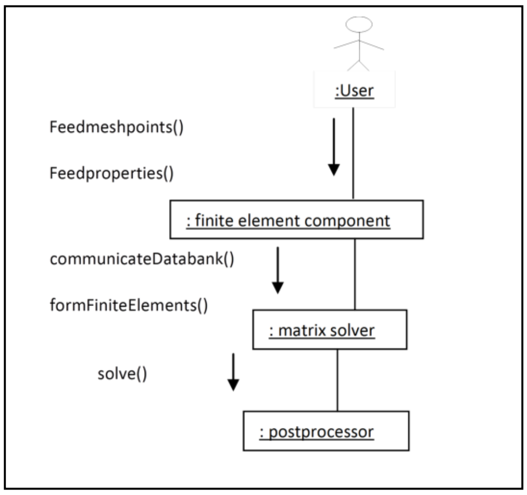

Fig. 2. Collaboration diagram for finite element software

Re-engineering Finite Element Code

UML diagrams [2] are the main modern software tool for analyzing and designing code, in this instance finite elements code. Class diagrams in UML are used to analyze and design interfaces between modules, a must to facilitate reuse and when components are put together [8]. Re-engineering or reverse engineering is the engineering (i.e., re-designing) of existing code which had been developed in an earlier era when there were few software standards – so called legacy code [13-18]. A powerful UML feature to re-engineer code is to give object oriented source code as input and generate the class diagrams that capture the software design underlying the code [19, 20]. This generated class diagram then would yield any mistakes in the code. The corrected class diagram can be used to recreate the code with corrections. This corrected code essentially does not give the detailed algorithms but rather the object oriented code with the headers and passing parameters for all functions [8].

For example, previously existing Java code of the class called Matrix shown in Fig. 1 is used to create the class diagram using ArgoUML^{TM} [8]. After creating the class diagram, access level modifiers of the fields "columns" and "rows" were changed from "private" to "protected". In ArgoUML^{TM}, facility is given to select a suitable access level modifier using "radio buttons". Then the forward engineering facility is used to create Java code, shown in Fig. 2, for the modified class diagram. The changes are shown in lines 3 and 4 in Fig. 1.

Reverse engineering can be used to convert from one language to another. Assume that source code in Java is to be converted to C++. After creating the class diagram using the existing Java code, reverse engineering is used to create C++ code. The generated code can be edited later to improve efficiency or to look simpler. When the code is generated in C++, header files with extension .h, will also be created. Such header files should be included in the source file. An attempt was made to convert a fully object oriented finite element software package written in C++ [8] to Java code using re-engineering and forward engineering facilities or techniques. In the Java code, the outline of the classes with property names and method names is generated. The body of the Java code needs to be filled. Examples are not displayed here because of the applicable space limitations.

```
PROGRAM TEST
  PRINT*,"Hello WELCOME FORTRAN TO C "
 END
```

Fig. 3. FORTRAN Test Program

```
/* Hello.f -- translated by f2c (version 19980831 for lcc-win32).
   You must link the resulting object file with the library:
 libf77.lib
*/
#include "f2c.h"
/* Table of constant values */
static integer c__9 = 9;
static integer c__1 = 1;
#line 3 ""
/* Main program */ MAIN__(void)
{
   /* Builtin functions */
integers_wsle(cilist *), do_lio(integer *, integer *, char *,
ftnlen),
    e_wsle(void);
   /* Fortran I/O blocks */
staticcilist io___1 = { 0, 6, 0, 0, 0 };
s_wsle(&io___1);
do_lio(&c__9, &c__1, "Hello WELCOME FORTRAN TO C ",
(ftnlen)27);
e_wsle();
return 0;
} /* MAIN__ */
/* Main program alias */ int test_ () { MAIN__ (); return 0; }
```

Fig. 4. C Code converted from the FORTRAN Test Program of Fig. 3

Sequence Diagrams in Re-engineering Finite Element Code

Sequence diagrams are an aspect of UML diagrams that give the sequence of operations that are being programmed [1, 2]. In a simple test of three commercial codes, not named here, it was found that the sequential operations of preprocessing, solving and post-processing were offered as parallel processes. Thus without preprocessing the problem, the solution could be attempted and without solving the problem post-processing could be invoked. This would not have happened if proper software engineering techniques had been employed in development with sequence diagrams. An interaction diagram is given in one of two forms: sequence diagrams or collaboration diagrams [1, 2]. A collaboration diagram for finite element method software is given in Fig.2.

Re-engineering FORTRAN Code

One of the most important aspects of reverse engineering would involve means of using legacy code written originally in FORTRAN since much of the legacy code is in FORTRAN.

Now facilities are available to convert FORTRAN to C and then C to Java or even FORTRAN directly to Java [20]. The authors tested this by converting programs from FORTRAN to C and C to Java. It is noted that

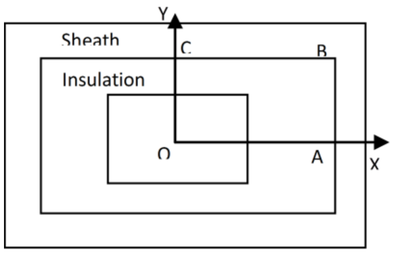

Fig. 5: Two-Conductor Cable

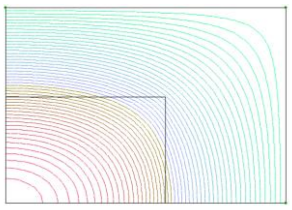

Fig. 6: Solution for Two-Conductor System

(a) Classes cannot directly be created from FORTRAN code using forward engineering. Therefore, to make it possible, FORTRAN code should be converted/translated first into Java code and it is thereafter that classes can be created, using re-engineering.

(b) Earlier, FORTRAN programs were written based on a functional approach to programming rather than an object oriented approach. Therefore even if FORTRAN code is converted to Java code, it will not be in object oriented design.

(c) In the conversion process, a given program is considered as input and it is more or less like an interpreter that translates the given source code into intermediate code; i.e. for example, the FORTRAN code of Fig. 3 is auto-translated using software freely available for that purpose [18-20] to the C code of Fig. 4 where very many statements are generated but the output which displays “Hello WELCOME FORTRAN TO C” is the same. Further, the generated C program is linked to some library files when it is run. Therefore when the generated C program is converted into Java code, the library files cannot be linked properly from Java platforms.

It was found that the problem arises because of input/output operations. Therefore, input/output operations in the converted C program are rewritten to suit the format of the C programming language. After this modification, the converted Java program works well.

The C program converted from FORTRAN shown in Fig. 4 was converted into Java successfully and the converted Java program contains 53 lines. Because of the limited space for this article, the converted Java program is not displayed here. In addition to that the FORTRAN program shown in Fig. 3 consists only of a single statement excluding the beginning and finishing lines and the converted code in C shown in Fig. 4 contains several lines; i.e. a single line in FORTRAN takes many lines in the converted code. Further, only a single class is generated in Java. A finite element program developed in FORTRAN was successfully converted into Java by the authors.

Though trivial, a simple problem allowing matrices of various sizes to be easily generated is taken for the purpose of comparing the execution times of programs written to solve an electromagnetic field problem. In our example, one program was written using the FORTRAN language and then it was converted into C and then that into a Java program. Also a fresh Java program was developed based on the design ideas taken from the converted Java program and the execution times of all the above programs are compared. In our sample problem, we wish to determine the magnetic field within a cable system. At the outer sheath, the reference magnetic vector potential A is zero. Using the symmetry inherent to the problem, the problem can be reduced to one fourth of the full problem. Fig. 5 describes the problem area and its nature. Instead of the whole region, the area OABC can be considered with suitable boundary conditions. Then triangular finite elements are formed for the domain OABC. Likewise, for the same problem different meshes are formed with different number of mesh points, triangles and unknowns and the problem solved by the different programs mentioned above and execution times are compared.

Table 1 shows the execution time comparison of the FORTRAN program, the C program (converted from the legacy FORTRAN program), the Java program (converted from the C program) and the Java program newly written using an object oriented approach based on the designs seen in the converted Java program. Further, in these programs, the profile storage scheme

Table 1: Time Comparison of FORTRAN program, C (Converted from FORTRAN) program, Java (Converted from C) Program and Java program (written) (on a machine with Intel(R) Core(TM) 2 Duo CPU E7300 @ 2.66GHz, 1024 RAM)

No. of points	**No. of Trian -gles**	**No. of Un-knowns**	**Time: FOR -TRAN** (Sec)	**Time : C (from FORT-RAN)** (Sec)	**Time: Java (from C)** (Sec)	**Time: Java (written)** (Sec)
36	50	25	0	0.0000	0.0000	0.0000
100	162	81	0	0.0000	0.0000	0.0000
196	338	169	0.0016	0.0000	0.0032	0.0000
324	578	289	0.0016	0.0000	0.0046	0.0030
484	882	441	0.0078	0.0000	0.0110	0.0064
676	1250	625	0.0156	0.0000	0.2219	0.0094
900	1682	841	0.0297	0.0000		0.0188
1156	2178	1089	0.0484	0.0000		0.028
1444	2738	1369	0.0750	0.0001		0.0376
1764	3362	1681	0.1219	0.0001		0.053
2116	4050	2025	0.1969	0.0002		0.072
2500	4802	2401	0.3266	0.0003		0.1
2916	5618	2809	0.4531	0.0004		0.128
3364	6498	3249	0.6078	0.0005		0.175
3844	7442	3721	0.8188	0.0008		0.2156
4356	8450	4225	1.1922	0.0012		0.275
4900	9522	4761	1.8781	0.0018		0.3374
5476	10658	5329	2.6297	0.0026		0.4532
6084	11858	5929	3.6156	0.0039		0.525
6724	13122	6561	5.1531	0.0055		0.6406
7396	14450	7225	6.9031	0.0073		0.8656
8100	15842	7921	9.0734	0.0092		0.9218
8836	17298	8649	11.3766	0.0115		1.4906
9604	18818	9409	14.1734	0.0146		3.85
10404	20402	10201	31.5703	0.0284		47.0312
11236	22050	11025	37.2578	0.0319		86.7968

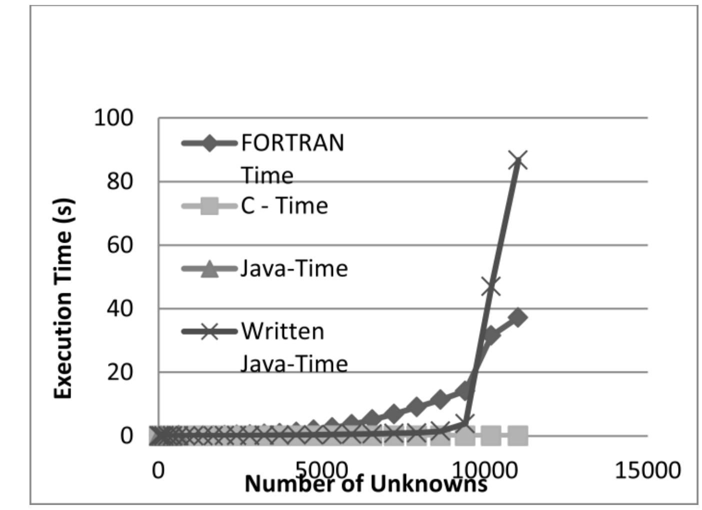

Fig. 7: Graph of execution time versus number of unknowns

is used to represent matrices because profile storage consumes less memory compared to a two dimensional array which will store all zero and nonzero matrix elements unlike the profile storage scheme.

At a point (for matrices above size of about 625x625) the converted Java program stopped working. In the converted Java program a two-dimensional array is automatically converted into a one dimensional array. Therefore more time is needed for calculations related to the conversion process. Further during the conversion process, some additional variables are used for calculations. Therefore more memory is used in the converted java program. Also in the converted program, since a one dimensional array is used, the memory location of each element is calculated and the memory size of each element is also used in the calculations. Suppose that there is an m x n two dimensional array (like a matrix A with m rows and n columns) and the address of A(i,j) is to be found. The way Java works, the memory location of A(i,j) is from:

$$\text{ADDRESSA}(i,j) = \text{SMA} + i*n*B + j*B \quad (1)$$

where ADDRESSA(i,j) is the memory address of element A(i,j); SMA is the starting memory address of array A; and B is the number of bytes to be used for the type of the array element. In Java 0 <= i < m and 0 <= j < n. For example if m = 5000; n = 5000; i = 4000; j = 2500 and B = 4, then

ADDRESSA(i,j) = SMA + 80010000. As the memory address is represented using integers, when the above ADDRESSA(i,j) exceeds the integer range used by Java or in some cases by the computer or the operating system, the program will be stopped; i.e., it will hang. Because of the above reason, the converted Java program could not be run with more than 625 unknowns in this work. In the newly written Java program, most of these unwanted address calculations and conversions are discarded. Therefore the freshly written Java program consumes much less time than the converted Java program in executing the program. Fig. 7 shows the comparison of execution times of the FORTRAN program, the C program (converted from FORTRAN), the Java program (converted from C) and the newly written Java program versus the number of unknowns. Once the Java program is developed, re-engineering is used to develop a class diagram which helps to write programs in any object oriented language. Also in the freshly written Java program, a matrix is represented as a vector of vectors. When the number of unknowns is increased, more calculations are involved in dealing with matrix elements. Therefore execution time went up drastically with unknowns.

Conclusions

This paper has investigated the reengineering of legacy finite element code for purposes of improvement and correction; use of modern object oriented features to obtain the best design and correct code from a software engineering perspective; and the running of the code on the internet by converting the code to Java. As a test, working finite element code authored in C++ has been reengineered to Java. This process recreates the class designs that went into the C++ code, thereby permitting an examination of the design to facilitate improvements to the C++ code itself or alternatively creating Java code using that design. FORTRAN code needs hardwiring since the concept of class does not exist in FORTRAN. Changing over to object oriented languages from FORTRAN involves designing the classes. A program to solve electromagnetic problems using the finite element method is written using FORTRAN and that program is converted into the C language and from C to the Java language using automatic tools. Then a Java program is written in an object oriented approach based on the Java program converted from the C program which in turn had been converted from the original FORTRAN program. The execution times of the all the above programs are compared.

As to be expected, C works best. Automatically converted Java programs fail because of issues with integers used for memory pointers in Java, but when such converted programs are used to discover the original design of the FORTRAN program which was converted, and used to rewrite a program from scratch in Java, significant improvements are realized.

Acknowledgement

Mr. Thiruchelvam Arudchelvam thanks his employer, Wayamba University of Sri Lanka, for doctoral study leave; University of Peradeniya Sri Lanka where he is a doctoral student; and Rensselaer Polytechnic Institute for facilities to work on his thesis as a Visiting Fellow.

References

[1] Bernd Bruegge and and H. Dutoit: *Object-oriented Software Engineering – Using UML, Patterns and Java,* 2nd edn. (Pearson-Pentice Hall, Upper Saddle River, NJ, 2004)

[2] Scott Ambler, *The Elements of UML 2.0 Style* (Cambridge Univ. Press, New York, 2005)

[3] G. Booch, R.A. Maksimchuk, M.W. Engle, B.J. Young, J. Conallen and K.A. Houston, *Object Oriented Analysis and Design with Applications* (Third Edn.) (Addison Wesley, Upper Saddle Riiver, NJ, 2007)

[4] S. R. H. Hoole and T. Arudchelvam: *Revue roumaine des sciences techniques - Série Électrotechnique et Énergétique* Vol. **54** (2011)

[5] Haiwei Wang, Geng Liu and Liyan Wu: *Materials Science Forum* Vols. 532-533 (2006) pp. 909-912

[6] H.M. Wang, T.Y. Wang and X.J. Guo: *Materials Science Forum* Vols. 626-627 (2009) pp. 687-692

[7] S.R.H. Hoole, A. Mascrenghe and K. Navukkarasu: Ref. 131, Proc. JAPMED’4 (2005) Ref. 131

[8] S.R.H. Hoole and T. Arudchelvam: Proc. *6th Japanese-Mediterranean Workshop on Applied Electromagnetic Engineering (JAPMED06)* (2009).

[9] Harry G. Schaeffer: *MSC/NASTRAN primer: Static and normal modes analysis* (Schaeffer Systems, 1982)

[10]B.L. Achee and D.L. Carver: *J. Systems and Software*, Vol. 39 (1997) pp. 179-194.

[11]J. Cheng, S-Y Kwak and H-Y Hwang: *Int. J. of Cast Metals Research*, Vol. 24 (2011), No. 3-4, pp. 238-242.

[12]K.R.C. Wijesinghe, M.R. Udawalpola and S.R.H. Hoole: *Digests of the IEEE CEFC*, 2002**.** Information on http://www.sciencedirect.com/science?_ob=ArticleURL&_udi=B6TGJ-4DTM097-2&_user=10&_rdoc=1&_fmt=&_orig=search&_sort=d&_docanchor=&view=c&_acct=C000050221&_version=1&_urlVersion=0&_userid=10&md5=0fb833de0c7b46d78c769aae708445bc

[13]Ira D. Baxter and Robert Larry Akers: *Proc. 20th IEEE International Conference on Software Maintenance* (2004) pp. 509-.

[14]M. Hanna, *Software Magazine*, Wednesday, September 1 1993.

[15]Pedro Sousa and Jürgen Ebert, *Proc. Fifth European Conference on software Maintenance and Reengineering* (IEEE Computer Society Press, 2001).

[16]A. De Lucia, Rita Francese,G. Scanniello and G. Tortora: *Software: Practice and Experience* Vol. 38(13) (2008) pp. 1333-1364

[17]K. Kontogiannis, C. Tjortjis and A. Winter (Eds.) *J. of Software Maintenance and Evolution – Research and Practice*, Vol. 21 (2009)

[18]Information on http://argouml.tigris.org/

[19]Information on http://www.visual-paradigm.com/product/vpuml/

[20]Information on http://www.netlib.org/java/f2j/

Applied Electromagnetic Engineering for Magnetic, Superconducting and Nano Materials
10.4028/www.scientific.net/MSF.721

Reverse Engineering Legacy Finite Element Code
10.4028/www.scientific.net/MSF.721.307