Teaching Object-Oriented Numerical Analysis

S.D. Rajan[1],
[1]Professor of Civil, Aerospace and Mechanical Engineering
Arizona State University
Tempe, AZ 85287-5306, USA

Abstract- While a vast majority of engineers do not write computer programs for a living, every engineer working on any facet of their job responsibilities, deals with data and the means of extracting meaningful information from the data. The backbone of such analyses is formed by numerical methods. A very powerful engineering and scientific tool can be developed when numerical methods are combined with object-oriented techniques. The development of such a framework is discussed in this paper, and the implementation details are presented in the context of a senior undergraduate and graduate course where object-oriented programming via C++ is tied to numerical methods such as the finite element analysis.

S.D. Rajan[1],
[1]Professor of Civil, Aerospace and Mechanical Engineering
Arizona State University
Tempe, AZ 85287-5306, USA

I. INTRODUCTION

Numerical methods constitute an important part of all scientific or engineering curricula. However, by themselves, numerical methods have limited applicability if they are not implemented as computer programs. The first high-level language that tackled this problem was FORTRAN that made a huge impact in the area of scientific and engineering computations in the 1950's [McJones, 2015]. To engineers, the language's syntax was a huge leap over machine language or assembly language. They were able to write and debug complex programs more efficiently, almost all of which incorporated some form of numerical computations. The important role that numerical methods play in engineering analysis and design is evident in the vast number of computer programs used in academia and the industry. With increasing sophistication of the analysis methods and tools, the enormous amount of data to be handled, the ready availability of computing tools and the need for robust, fast, easy-to-use computer programs, one cannot look at numerical analysis and computer programming as two different entities. The interesting point, however, is that, with the advent of data mining, numerical methods are back on the scene of information technologies [Besset, 2015]. Today, we are routinely solving problems that would be impossible to attack with the help of pen, paper and brain alone [Arge et al., 1999].

In this paper, we examine specifics of an upperdivision/graduate course CEE432/CEE532 Developing Software for Engineering Applications that is taught to senior students and primarily, graduate students. The course has three course objectives - (1) understand the process of engineering software development, (2) learn the concepts associated with (i) object-oriented problem solving, (ii) algorithm development, implementation and debugging, and (iii) resource allocation and performance tuning in engineering software, and (3) learn how to apply object-oriented techniques towards the development of engineering analysis and design tools such as finite element analysis software system and/or geometric modelers. The primary pre-requisites to the course include mathematics (differential equations, linear algebra, numerical analysis), engineering mechanics (statics, dynamics, strength of materials, engineering materials), structural analysis, and knowledge of a high-level programming language at an undergraduate functional level.

In this concept rich, application-oriented course, student assessment poses strong challenges.

The course is divided into two halves. In the first half, C++ is introduced at a sufficiently high level so that students can develop, write and debug relatively complex and useful programs. At the very early stage, numerical analysis and C++ are intertwined without formally defining objects – understanding floating point representation and operations, using modular program development to write functions for root finding, numerical differentiation and integration while understanding the nuances of numerical computations. However, examples are presented where even in simple examples, objects are used – *string* class to handle character strings, *iostream* library to handle input and output, input/output manipulators for formatting output, and user-defined vector and matrix classes to carry out matrix operations. At this stage, the benefits of using an Integrated Development Environment (IDE) are introduced along with the use of an interactive debugger as a powerful aid in software development. The concept of classes and objects is introduced next. After briefly discussing pointers, a strong case is made for dynamic memory allocation. Two template classes are developed – a vector class to store and manipulate vectors, and a matrix class to store and manipulate two-dimensional arrays. Students at this stage of the semester are ready to plan, develop and test a template matrix toolbox that has a host of matrix-related functionalities – array addition, subtraction and multiplication, transpose, dot and cross products, vector norms, solution to linear algebraic equations using Cholesky factorization with full, banded and other more efficient storage schemes. Finally, the first part ends with a study of advanced concepts such as file handling, and data abstraction concepts such as inheritance and polymorphism. Resources for the first part of the course include electronic notes on numerical analysis-based object-oriented programming and an extensive library of functions, classes, and sample programs [Rajan, 2015] as will be illustrated throughout the paper.

The second part of the course closes the learning loop and deals with the implementation of object-oriented numerical analysis in the context of finite element analysis including rudimentary geometric modeling. At every stage, the benefits of object-oriented programming concepts are illustrated – code reuse, breaking a complex problem into a collection of smaller manageable pieces, developing robust, user-friendly and efficient classes (data encapsulation), organizing the collection into a hierarchical structure to facilitate inheritance, carrying out unit testing and other functionalities such as parsing data files.

The rest of the paper is divided into three parts. In the next section, we look at one approach to developing object-oriented numerical methods. This is followed by a section where we discuss the five step process to developing software in general and how the process is used in the class for the development of finite element program for three-dimensional truss and planar frame analyses. Finally, the last section summarizes the paper, looks at lessons learnt and plans for improving the course.

II. COMBINING NUMERICAL ANALYSIS AND OBJECT-ORIENTED CONCEPTS

The fundamental objective of the marriage between numerical methods and object-oriented programming is to create reusable code. Reusable code is attractive to use and leads to improved productivity if the code has a simple interface, is robust (adequate error checks) and is as efficient as possible. Since the second part of the course is focused on linear, elastic, small displacement and strain finite element analysis using truss and beam finite elements [Rajan, 2001], the list of required numerical methods is limited. Fig. 1 shows the collection of classes that meets the requirements.

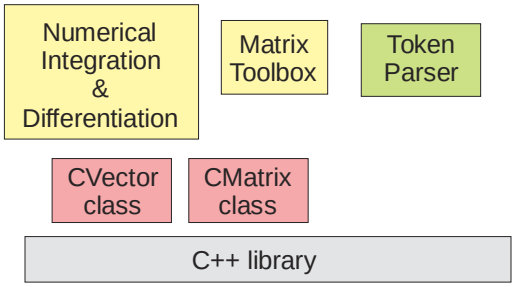

Fig. 1 A hierarchical scheme showing a collection of classes forming an object-oriented numerical library

The two foundational classes are the vector and matrix classes. These classes are template classes with a simple interface. For example,

```
CVector<float> Coor(NumNodes);
CMatrix<double> K(nDOF, nDOF);
```

shows the declaration for a vector and matrix whose sizes are known only at runtime. Indexing starts at 1 rather than 0, and in the debug mode, the index values are checked to see if they are out-of-bounds. It should be noted that these two features were specifically made a part of the classes to make them easy to use (engineers do not start counting from zero) and indexing errors is the most common error in the use of arrays. Furthermore, these classes are also designed as rudimentary containers to facilitate the storage of objects. The matrix toolbox is built on top of the vector and matrix classes with the functionalities discussed in the previous section including the ability to track floating point operations to compute FLOPS. Finally, the numerical integration and differentiation classes round off the numerical library.

In the first major project in the course, the students develop and test a matrix template toolbox. For most of the students, this is their first exposure to systematic development of a computer program that is non-trivial and is useful for all courses where numerical analysis is a focus. For example, having declared a matrix toolbox object, they can write a simple code to test components of the toolbox such as matrix

multiplication or solution of linear algebraic equations using Cholesky factorization.

```
    CMatToolBox<double> MTBDP;
…
    // matrix multiplication test
    if (!MTBDP.Multiply (dMA, dMB, dMC))
..
    double TOL = 1.0e-6;
    if (!MTBDP.LDLTFactorization (dMCoef, TOL))
…
    if (MTBDP.LDLTSolve (dMCoef, dVSol, dVRHS))
```

Students also learn that manipulating floating point numbers poses a challenge and that they need to write functions that carry out logical comparisons between two values of scalars and arrays unlike other data types. For example, a function like

```
bool IsEqual (const double d1, const double d2,
    double TOL = TOLDEF);
```

is needed to check if two (seemingly equal) double values are equal to each other or not. Finally, a C++ token parser class is included in the library to facilitate reading and interpreting external text files that contain model data. For example, the parser class is able to break the components in a text input file allowing the client code to interpret and store the data with a correct input, or flag an incorrect input. For example, a file that contains the following data in an external file

```
*nodal coordinates
1 0.0 0.0
2 6.0 0.0
3 10.0 0.0
```

can be parsed and stored using the following code segment

```
// read nodal coordinates
CVector<float> fVC(NDIM);
if (!Parse.ReadNextLine (m_FileInput, m_nLineNumber, szInputString,
                         MAXCHARS, szComment))
    IOErrorHandler (INVALIDINPUT);
for (i=1; i <= m_nNodes; i++)
{
    if (!Parse.ReadNextLine (m_FileInput, m_nLineNumber, szInputString,
                             MAXCHARS, szComment))
        IOErrorHandler (INVALIDINPUT);
    else
    {
        std::istringstream szFormatString (szInputString);
        szFormatString >> nTag >> fVC(1) >> fVC(2);
        if (szFormatString.fail() || szFormatString.bad())
            IOErrorHandler (INVALIDINPUT);
    }
    if (nTag <= 0 || nTag > m_nNodes)
        IOErrorHandler (NODENUMBER);
    m_NodalData(nTag).SetCoords (fVC);
}
```

III. THE FIVE STEP PROCESS TO SOFTWARE DEVELOPMENT

Computer scientists point to the waterfall model as one way of developing software [Wikipaedia, 2015]. With students who have never been engaged in the software development process, a simpler but equally structured approach can be used as shown in Fig. 2. When the requirements are known exactly and the final product is predictable, this approach leads to a manageable task for students either working alone or in small groups.

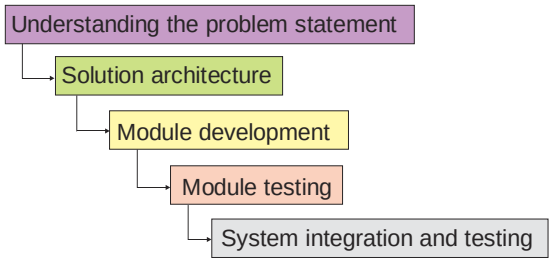

Fig. 2 Course software development framework

To illustrate the course software development framework (CSDF), two software development projects are used in the second half of the course. The first is development of a space truss finite element analysis program as shown in Fig. 3. The source code for the project is given to the students and the project requirements are to incrementally add and enhance the capabilities of the program including integrating their matrix toolbox into the system.

CEE 532/MAE541 Developing Software for Engineering Applications
Project 2: Space Truss Analysis Program

1.0 Project Statement: Develop a program to carry out space truss analysis. You may use the planar truss program discussed in class and posted on the web site as a starting point. The program must have the following capabilities.

(1) There should be no artificial restriction on the size of the problem that can be solved. In other words, use dynamically allocated arrays. Use arrays only when required.
(2) ***The input file format (Section 2.0) must be strictly followed***. Assume that the input file is created using consistent units (for length, force and temperature). Program must detect input errors and print out meaningful error messages.
(3) The loading on the truss can be due to (a) nodal forces, (b) nodal displacements and (c) nodal temperature changes.
(4) The program must compute (a) nodal displacements, (b) element strain, (c) element stress, (d) element force, and (e) support reactions. These computed quantities must be written in a tabular form (see Section 3.0). You must also compute the relative and absolute error norms and write them to the output file.
(5) You must use the matrix toolbox that you developed in Project 1.
(6) The program must ask only for the input and output file names. It should indicate that the program has been successfully executed or display the appropriate error message. Do not display any debugging outputs.

Fig. 3 Space truss program specifications (only first section of the document is shown here)

The program specifications document is used as a starting point for the CSDF discussions.

Step 1. Understanding the problem statement
The students read the problem statement and develop a document that identifies the capabilities of the program including the overall solution algorithm, the input model, and the contents of the output file.

Step 2. Solution architecture
Based on the program specifications and the given planar truss program, students are asked to draw the class diagram for the entire program. They identify the existing capabilities and the

missing capabilities. They then break the overall class diagram into standalone modules.

Step 3. Module development
At this stage, they are ready to use a bottom-up approach to define for each module its capabilities, the input required to drive the module, and the output state that the module creates. They take the required steps from the overall algorithm (Step 1) and are ready to write a detailed algorithm for each module. At this stage they are ready to either write or modify the source code.

Step 4. Module testing
Steps 3 and 4 are done iteratively to ensure that each module performs as stated or two or more modules interact correctly. The process of creating unit tests gives the students a new perspective on testing. They understand the differences between testing methods versus testing classes versus testing modules.

Step 5. System integration and testing
Finally they are ready to integrate the modules and carry out tests to ensure that the entire program is working correctly. A sample main program is shown in Fig. 4.

```
/*******************************************
Planar Truss Analysis Program
Copyright(c) 2000-15, S. D. Rajan
All rights reserved

Introduction to Structural Analysis and Design
Object-Oriented Numerical Analysis
*******************************************/
#include "truss.h"

int main (int argc, char* argv[])
{
    CTruss MyTruss; // the one and only truss

    // show program banner
    MyTruss.Banner (std::cout);

    // open input and output files
    MyTruss.PrepareIO (argc, argv);

    // read truss model
    MyTruss.ReadTrussModel ();

    // analyze
    MyTruss.Analyze ();

    // close input and output files
    MyTruss.TerminateProgram ();

    return 0;
}
```

Fig. 4 Main program for the planar truss program

In the second software development project (development of a planar frame analysis program using beam finite elements), the program specifications are more loosely defined in order to encourage the students to use more advanced concepts such as inheritance and polymorphism and experience the advantages of those concepts [Rajan, 2015].

IV. INSTRUCTION AND ASSESSMENT CHALLENGES

Both instruction and assessment modes are challenging in this course. In the pre-requisites to this course, students are exposed to the *flipped classroom* style teaching technique [Sirigiri and Rajan, 2015]. This style is continued in this course but with some changes. Microsoft Visual Studio [Microsoft, 2015] is used and the students bring their laptops to class where they practice not only writing the entire code but take an existing code and enhance its capabilities. This experience and knowledge makes it possible to have in-class, open-book and notes, quizzes where the students write very focused computer programs during the first part of the course. At a later stage, the emphasis turns to the development of project-oriented exercises where the bulk of the discussions take place in class (with some on the course discussion board) and almost all of the work is done by the students outside of the classroom.

V. CONCLUSIONS

While object-oriented numerical analysis is not a panacea for solving engineering problems, a properly structured course can be used to meet the course objectives listed earlier. With proper in-class discussions, examination of case studies and project-oriented assignments, students understand the process of engineering software development, learn why object-oriented numerical analysis leads to code reuse and helps in code maintenance, and learn how to apply object-oriented techniques towards the development of non-trivial engineering analysis and design tools.

Here is a brief summary of the lessons learnt – (1) students learn to program in several ways and using the right software development tools pays off in the long run, (2) there are no better ways of showing that object-oriented numerical analysis helps scientific and engineering programming than to discuss the development and implementation of non-trivial computer programs that “would be impossible to attack with the help of pen, paper and brain alone”, (3) writing computer programs is as much an art as it is science and that students should be given the freedom to explore their creativity, and (4) students excel only when challenged (by this we mean that we need to push against the resistance to change, and resistance to self-learning or discovery and resistance to performing as engineering professionals).

REFERENCES

E. Arge, A.M. Bruaset and H.P. Langtangen, “Object-Oriented Numerics”, Numerical Methods and Software Tools in Industrial Mathematics, M. Daehlen and A. Tveito (eds.), pages 7-26. Birkhauser, 1997.

D.H. Besset, Object-Oriented Implementation of Numerical Methods An Introduction with Smalltalk, https://github.com/SquareBracketAssociates/NumericalMethods, 2015.

P. McJones, The History of FORTRAN and FORTRAN II, http://www.softwarepreservation.org/projects/FORTRAN, accessed September 7, 2015.

Microsoft, https://www.visualstudio.com/en-us/visual-studio-homepage-vs.aspx, accessed September 9, 2015.

S.D. Rajan, Introduction to Structural Analysis & Design, Wiley, 2001.

S.D. Rajan, Object-Oriented Numerical Methods via C++, electronic notes, Arizona State University, 2015.

M. Sirigiri and S.D. Rajan, "A Flipped Classroom Approach to Teaching Engineering Mechanics Courses", Journal of Engineering Education Transformations, 28:4, 109-113, April 2015. Also 2nd International Conference on Transformations in Engineering Education, Bangalore, India, January 2015.

Wikipaedia, https://en.wikipedia.org/wiki/Waterfall_model, accessed September 9, 2015.