Abstract—Object oriented design and software engineering are two major areas in software development. Object oriented design is very useful for solving electromagnetic problems, as classes can be utilized and reused for solving different kinds of problems by just integrating the classes which are already developed for solving other problems. Further, developing a suitable user interface will make it easy to the users depending on the nature of the problems and the level of the users. Software engineering is used to utilize, improve and adopt the legacy finite element codes. Software engineering principles were not properly implemented when the legacy finite element codes were developed. In early days, those legacy codes were developed in an ad-hoc basis. Much of those codes were written in FORTRAN programming language and modern software developers are facing difficulties in understanding and modifying those codes according to the present needs. This paper proposes and analyses a method to utilize and adopt legacy finite element codes and proposes a design which is ever useful for software engineers in the future.

Index Terms—Component, finite element software, FORTRAN, object oriented design.

I. OBJECT ORIENTED PROGRAMMING

Object oriented program design and development is now a standard in software engineering practice. Object oriented design attempts to make programs more closely model the way people think about and deal with the real world. In the older styles of programming, a programmer who is faced with some problems must identify a computing task that needs to be performed in order to solve the problem. Programming then consists of finding a sequence of instructions that will accomplish that task. But at the heart of object-oriented programming, instead of tasks, we find objects—entities that have behaviours that hold information, and that can interact with one another. Programming consists of designing a set of objects that model the problem at hand. Software objects in the program can represent real or abstract entities in the problem domain. This is supposed to make the design of the program more natural and hence easier to get right and to understand [1].

The principal advantage of object oriented design is reusability. It is ideal for describing autonomous agents so that values inside a method are private unless otherwise so provided—that is encapsulation makes programming neat and less error-prone in unexpected situations. Further, each subclass inherits the variables and methods of its super-class and that is called inheritance. Another notable point in object oriented design is polymorphism which facilitates the assignment of different meanings or usages to something in different contexts – specifically, to allow an entity such as a variable, a function, or an object to have more than one form. For example, there may be many functions of the same name and only the number of parameters, type of parameters or the order in which they appear will be different. When the function which has several forms is activated, the most suitable function is automatically chosen, i.e. selected, by the program depending on the number of parameters, type of parameters or the order of parameters.

Moreover, modern software practice encourages component-based design, giving users the freedom and choice to put together, i.e. glue together, different methods to custom-make the program. This is closely tied up to object oriented programming because of the concept of encapsulation and the user need to be bothered only with the functionality and not what is going on inside a method. In component-based design, each component is constructed as a class or a collection of classes. Therefore, object oriented design should be used in order to develop component based software packages. Depending on the needs of the user, suitable components are chosen and put together for the program.

Most of the programs we have for field computation using the finite element method, our legacy codes, were developed in an era preceding object oriented concepts. Much of this is often used with shells in the modern environments calling the old FORTRAN programs such as NASTRAN [2]. Even though some recent efforts have used object oriented concepts to develop finite element programs from scratch, their products remain in the private, licensed domain where we have no access. Component-based design was not even attempted. Reverse and forward engineering transformations are very useful in the process of adapting and improving finite element legacy code.

II. COMPONENT-BASED SOFTWARE

Component-based software engineering provides an opportunity for efficient software with better quality and increased productivity in software [3], [4]. Component-based software development facilitates software reuse and promotes quality and productivity. Such a building-block approach has been increasingly adopted for software development, especially for large-scale software systems. Much work has been devoted to developing
A component is developed independently of other components. Therefore, it is easy for a programmer to test and maintain the program. When an error in a component is reported, correcting the error of that particular component is enough and correcting the whole application is not required. Further, a particular component can be connected to any number of applications. In component-based software development, a collection of software components or library of components is integrated together in the system. When a problem is given to a programmer, first suitable components have to be decided by the programmer. If a component for a particular task is not available, then a new component should be developed and integrated with the existing collection of components. If all components are available, then a program is built by connecting all those components chosen and then a suitable user interface is developed by the programmer to simplify the task of the end user (Fig. 1).

III. COMPONENT-BASED FINITE ELEMENT METHOD SOFTWARE

Finite element method (FEM) software in electromagnetics and many other fields has grown out of huge engineering corporations like NASA and individual research laboratories. Early finite element programs were developed using FORTRAN programming language and focused on the solution of problems – something that needed to be computed in our context – and engineers from both corporate R&D and universities came up with the necessary software. The software failed to be put through the now mandatory lifecycle with planning, user requirements, object design, analysis, development and testing [8]. Such software is not always of the best design. Worse, because this legacy software is extensive and all but practically impossible to rewrite in the newer languages, it is continued to be in use with shells calling the old codes. A shell program is a program written to execute a specific task as the interpreter of user commands. In other words, shell programs are activated as user commands one at a time. Extensive scientific and numerical calculations are covered by these legacy codes. Because of not having been put through the software engineering design principles, these legacy codes cannot be discarded or rewritten as new modules as a result of the length of the codes. And to write new codes in another programming language, it is necessary to understand the existing legacy code. That is practically a tedious task, both complex and lengthy. Therefore, it is necessary to find a new method to avoid these kinds of practical problems. Further, it is necessary to find ways of using legacy codes in a suitable way in conformity with modern software standards. Previous works carried out by pioneers are in [9]-[29].

IV. SOFTWARE ENGINEERING APPROACH FOR DESIGNING FINITE ELEMENT METHOD SOFTWARE

Software engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software, and the study of these approaches; that is, the application of engineering to software [30]. In other words, software engineering is a study of the application of engineering concepts for developing software systems or projects. In software development, normally software engineering principles are applied to get a good outcome.

In this work, component-based finite element software is developed from legacy codes written in the FORTRAN programming language using software engineering principles. Component-based finite element software is suggested for better software design with the following features:

1) Software is composed of components and components are represented by classes/objects.
2) A particular component is allocated for a particular task.
3) After requirements elicitation, the developer should be able to pick up suitable available components and glue them together during the development process.
4) For a particular task, if a component is not available then that component is developed and integrated with the existing system of components so that it is available for future use.
5) A suitable user interface is properly designed so that all kinds of users may interact with the system and users should be able to pick up suitable components to suit an instance of a problem; i.e. the user is given an option to
choose components to suit a given problem at a particular instance.

Software engineering principles are applied during the development process of the suggested component-based finite element software so that the outcome of the process may widely be used, extendable and user friendly. Further, forward engineering and reverse engineering are used to convert a program written in one object oriented programming language into another desired object oriented programming language as shown in Fig. 2.

V. FORTRAN TO OBJECT ORIENTED DESIGN

The conversion process of finite element programs written in C and FORTRAN programming languages into an object oriented language is given in Fig. 3. The purpose here is to convert finite element programs written in FORTRAN and C programming languages into programs in an object oriented language. First, finite element programs written in FORTRAN are converted into programs in C. The details of this conversion and those issues related to the conversion are discussed in [31]. Then the C programs are converted into Java programs. Those Java codes are used to develop UML diagrams - especially class diagrams using reverse engineering. Again those class diagrams are used to create codes in any possible object oriented language such as C++, Java, etc. Therefore, this procedure gives a general idea to convert legacy finite element codes into an object oriented language. Further, suppose a new object oriented language is introduced, it would be easy for software engineers to develop codes using the available class diagrams. In this study, those finite element codes are converted into Java and C++ using reverse engineering and forward engineering [31], [32]. Further, performances of those finite element programs written in FORTRAN language, the relevant programs converted into C and the relevant programs converted into Java are compared and reported [33].

Once the UML diagrams for the design are ready, the whole systems can even be modified and new designs can be suggested. The codes for the new design in a suitable object oriented language can be derived using forward engineering. Keeping the UML diagrams of the design would help the software engineers to understand the system easily. Once the FORTRAN programs are converted into an object oriented design, it would be easy for the software engineers to redesign, modify and/or utilize those designs. Further, UML diagrams can be used to develop codes in a desirable object oriented language.

VI. RESULTS AND ANALYSIS

![Graph of execution times of FORTRAN, C and Java programs vs. number of mesh points of Program 1.](image)

Fig. 4. Graph of execution times of FORTRAN, C and Java programs vs. number of mesh points of Program 1.
Three different FORTRAN programs which use the finite element method were converted into C and Java. The performances of those three programs are compared based on the execution time. Mesh points are changed in the programs and checked the execution times. Graphs of execution times of FORTRAN, C and Java programs Vs. number of mesh points of 3 different programs are shown in Fig. 4, Fig. 5 and Fig. 6.

![Graph of execution times of FORTRAN, C and Java programs vs. number of mesh points of Program 2.](image)

![Graph of execution times of FORTRAN, C and Java programs vs. number of mesh points of Program 3.](image)

VII. CONCLUSION

In this work,
1) Finite element programs are converted into C and Java programs.
2) Converted C programs are also converted into Java programs.
3) Issues in the conversion process are reported [32].
4) UML class diagrams are created using converted Java programs.
5) From class diagrams, codes are again developed in C++ and Java and their validity is checked.
6) The class diagrams are modified and checked.
7) When there is a new problem to be solved using the finite element method, the classes are used in different ways. Only the user interface is changed according to the need of the user.
8) Using statistical analysis, the performances of codes written in FORTRAN, codes converted into C and codes converted into Java are compared and reported [33].
9) The components-based finite element methods software development method is discussed [34].

ACKNOWLEDGMENT

Thiruchelvam Aruduchelvam thanks University of Bath (UK) and RPI (US) for providing facilities to carry out the research work related to this paper. He also thanks the University of Peradeniya for their support to carry out this research work. Further, he thanks Wayamba University of Sri Lanka for the study leave and other moral support rendered.

REFERENCES
