Component-Based Design from Finite Element Software Written in the FORTRAN Language

T. Arudchelvam, S. Ratnajeevan H. Hoole, and Janaka Wijayakulasooriya

***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

Manuscript received September 22, 2013; revised November 15, 2013.

T. Arudchelvam Wayamba is with University of Sri Lanka, Kuliyapitiya (e-mail: tarudchelvam@gmail.com).

S. R. H. Hoole is with Michigan State University, Michigan, USA (e-mail: SRHHoole@gmail.com).

J. Wijayakulasooriya. is with University of Peradeniya, Sri Lanka (e-mail: jan@ee.pdn.ac).

infrastructure for the construction of component-based software [5]. The aims of component-based software development are to achieve multiple quality objectives, such as interoperability, reusability, evolvability, buildability, implementation transparency and extensibility, to facilitate fast-paced delivery of scalable evolving software systems. A Component-based software system often consists of a set of self-contained and loosely coupled components allowing plug-and-play [5]-[7].

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).

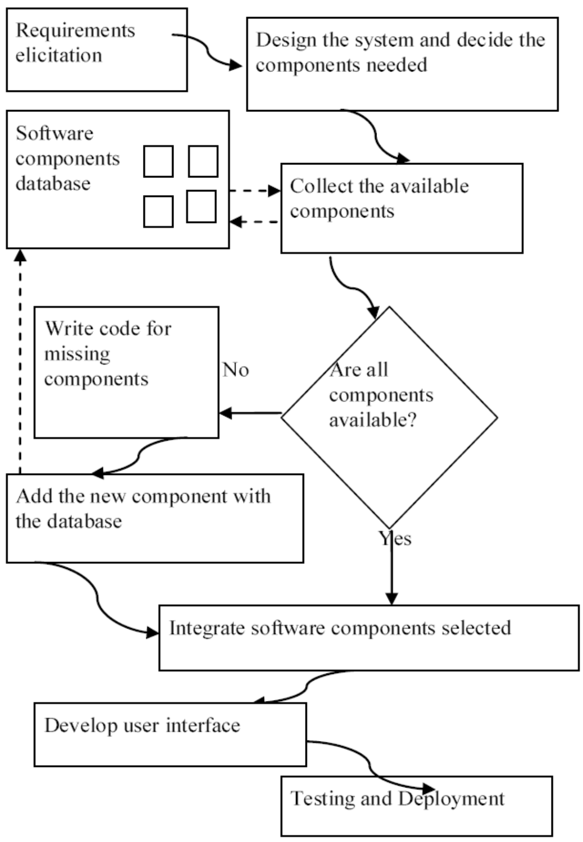

Fig. 1. Construction of finite element field computation software package using software components.

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 programing 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.

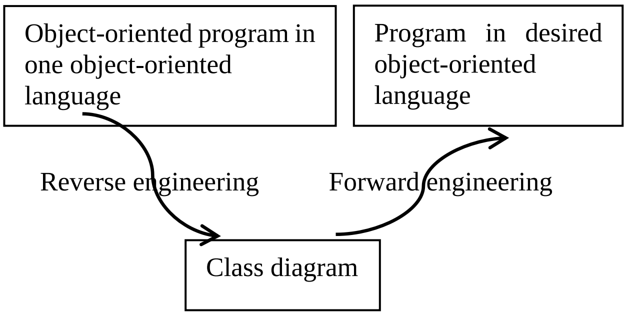

Fig. 2. Conversion process of an object oriented program from one object oriented language into another object oriented language.

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.

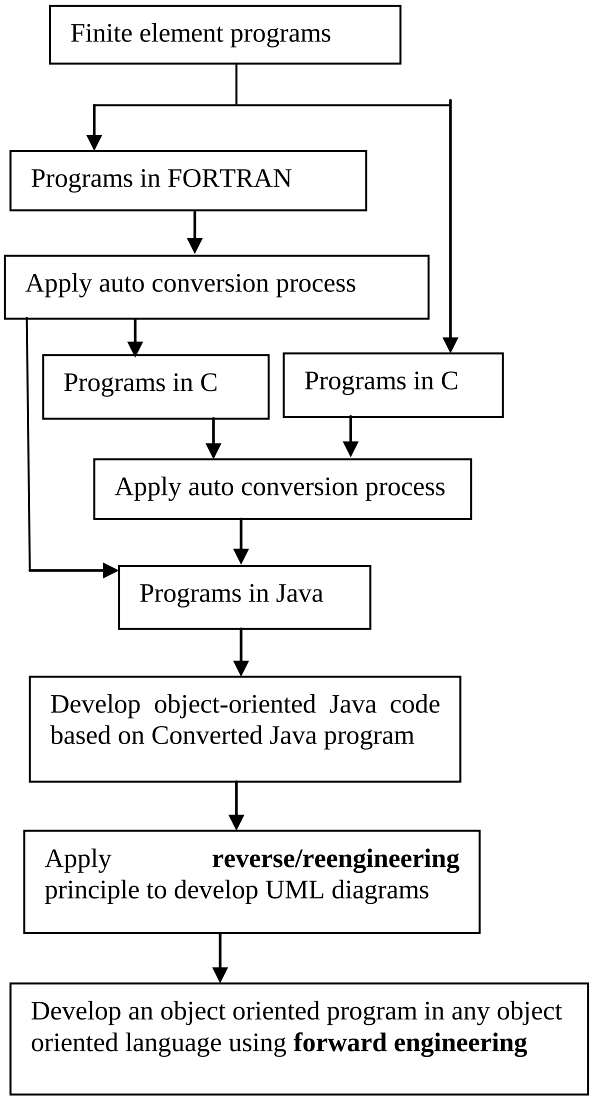

Fig. 3. Conversion of finite element programs into an object oriented programming language from legacy codes written in FORTRAN or C.

VI. RESULTS AND ANALYSIS

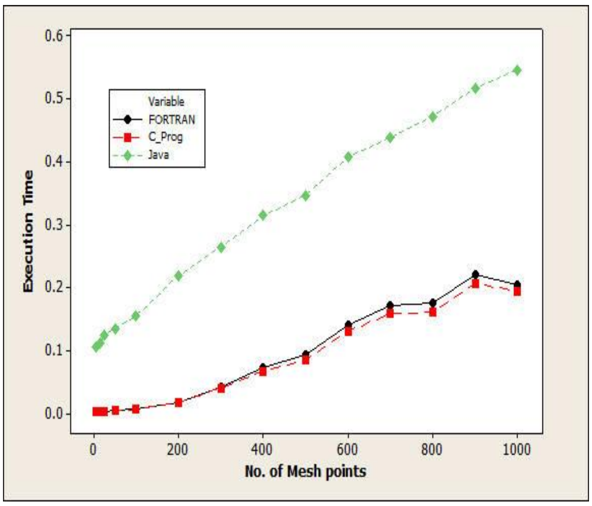

Fig. 4. Graph of execution times of FORTRAN, C and Java programs vs. number of mesh points of Program 1.

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.

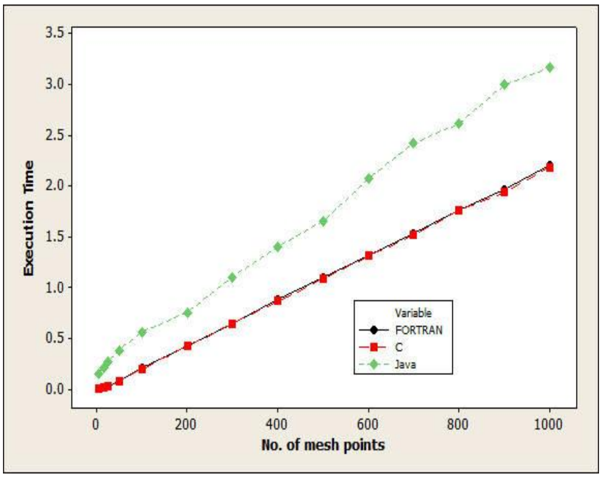

Fig. 5. Graph of execution times of FORTRAN, C and Java programs vs. number of mesh points of Program 2.

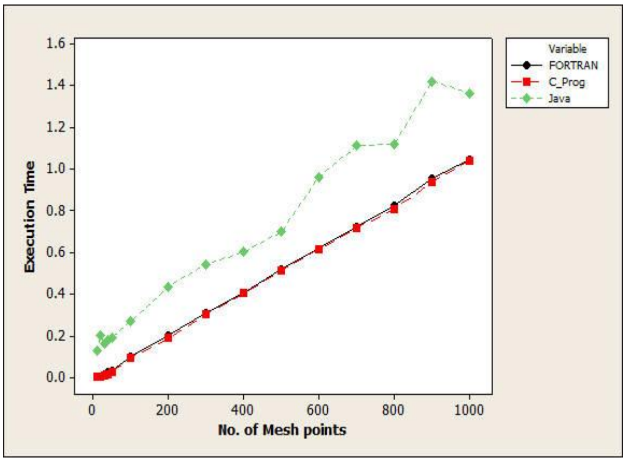

Fig. 6. Graph of execution times of FORTRAN, C and Java programs vs. number of mesh points of Program 3.

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 Arudchelvam 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

[1] A. Pillay. (November 01, 2009). Object-Oriented Programming. [Online]. Available: http://math.hws.edu/eck/cs124/downloads/OOP2_from_Univ_KwaZulu-Natal.pdf
[2] H. G. Schaeffer, "MSC/NASTRAN primer: Static and normal modes analysis," *Schaeffer Analysis*, 1982.
[3] X. Wu and M. Woodside, "Performance modeling from software components," in *Proc. the Fourth International Workshop on Software and Performance*, 2004, pp. 290-301.
[4] K. S. Jasmine and R. Vasantha, "Design based performance prediction of component based software products," *World Academy of Science, Engineering and Technology*, vol. 30, pp. 266-269, June 2007.
[5] Y. Wu, D. Pan, and M. Chen, "Techniques of maintaining evolving component-based software," in *Proc. International Conference on Software Maintenance*, 11-14 Oct. 2000, pp. 236-246.
[6] C. Szyperski, "Component technology - what, where, and how?" in *Proc. 25th International Conference on Software Engineering*, May 3-10, 2003, pp.684- 693.
[7] V. L. Narasimhan and B. Hendradjaya, "Some theoretical considerations for a suite of metrics for the integration of software components," *Information Sciences*, vol. 177, issue 3, pp. 844-864, 1 February 2007.
[8] S. R. H. Hoole and T. Arudchelvam, "A formal uml reliant software engineering approach to finite element software development for electromagnetic field problems," in *Proc. 6th Japanese-Mediterranean Workshop on Applied Electromagnetic Engineering (JAPMED06)*, July 2009.
[9] NASTRAN. [Online]. Available: http://en.wikipedia.org/wiki/Nastran
[10] SALOME: The Open Source Integration Platform for Numerical Simulation. [Online]. Available: http://www.salome-platform.org/home/presentation/overview/
[11] A. Ribes and C. Caremoli, "Salome platform component model for numerical simulation," in *Proc. 31st Annual International Computer Software and Applications Conference (COMPSAC 2007)*, vol. 2, July 2007, pp. 553-564.
[12] S. Ma, Y. Maréchal, and J. L. Coulomb, "Methodology for an implementation of the STEP standard: a Java prototype," *IEEE Trans. Magn.*, vol. 36, no. 4, pp. 1664-1668, July 2000.
[13] S. Ma, Y. Maréchal, and J. L. Coulomb, "Methodology for an implementation of the STEP standard: a Java prototype," *Advances in Engineering Software*, vol. 32, issue 1, pp. 15-19, January 2001.
[14] S. Ma, Y. Marechal, and J. L. Coulomb, "A finite-element 3-D magnetostatic solver using STEP data," *IEEE Transactions on Magnetics*, vol. 38, No. 2, part I, pp. 1097-1100, March 2002.
[15] S. Ma, Y. Maréchal, and J. L. Coulomb, "Methodology for an implementation of the STEP Standard: A Java prototype," in *Proc. Int. Conf. Compumag*, Sapporo, Japan, 1999.
[16] R. Ait-Sadi, B. F. Colyer, C. R. I. Emson, J. Simkin, and J. V. Maanen, "A 2-D and axisymmetric finite-element environment based upon STEP type database," *IEEE Trans. Magn.*, vol. 30, pp. 3622-3624, 1994.
[17] D. Rodger, P. J. Leonard, and H. C. Lai, "Surface elements for modelling 3D fields around thin iron sheets," *IEEE Transactions on Magnetics*, vol. 29, no. 2, pp. 1483-1486, March 1993.
[18] S. H. L. G. Bisson, P. J. Leonard, D. Rodger, and C. Leyden, "Finite element analysis of transient electromagnetic heating effects in three dimensions," *IEEE Transactions on Magnetics*, vol. 29, no. 1, pp. 1102-1106, Jan. 1993.

[19] P. J. Leonard, H. C. Lai, R. J. Hill-Cottingham, and D Rodger, "Automatic implementation of cuts in multiply connected magnetic scalar regions for 3D eddy current models," *IEEE Transactions on Magnetics*, vol. 29, no. 2, pp. 1368-1371, March 1993.

[20] K. R. C. Wijesinghe, M. R. Udawalpola, and S. R. H. Hoole, "Towards object oriented finite element preprocessors exploiting modern computer technology," *Journal of Materials Processing Technology (Elsevier)*, vol. 161, pp. 247-252, 2005.

[21] M. Dolenc, "Developing extendible component-oriented finite element software," *Advances in Engineering Software*, vol. 35, issues 10-11, pp. 703-714, October-November 2004.

[22] J. Peng and K. H. Law, "Building finite element analysis programs in distributed services environment," *Computers & Structures*, vol. 82, issue 22, pp. 1813-1833, September 2004.

[23] O. V. Estorff, "Coupling of BEM and FEM in the time domain: Some remarks on its applicability and efficiency," *Computers & Structures*, Elsevier, vol. 44, issues 1-2, pp. 325-337, 3 July 1992.

[24] K. L. Leung, P. B. Zavareh, and D. E. Beskos, "2-D elastostatic analysis by a symmetric BEM/FEM scheme," *Engineering Analysis with Boundary Elements*, vol. 15, issue 1, pp. 67-78, 1995.

[25] D. C. Rizos and Z. Wang, "Coupled BEM–FEM solutions for direct time domain soil–structure interaction analysis," *Engineering Analysis with Boundary Elements*, vol. 26, issue 10, pp. 877-888, December 2002.

[26] X. Zhang, "Coupling FEM and discontinuous BEM for elastostatics and fluid–structure interaction," *Engineering Analysis with Boundary Elements*, vol. 26, issue 8, pp. 719-725, 1 September 2002.

[27] S. P. V. Broeh, H. Zhou, and M. J. Peters, "Computation of neuromagnetic fields using finite-element method and Biot-Savart law," *Journal of Medical and Biological Engineering and Computing*, vol.34, no. 1, pp. 21-26, January 1996.

[28] A. R. M. Rao, "MPI-based parallel finite element approaches for implicit nonlinear dynamic analysis employing sparse PCG solvers," *Advances in Engineering Software*, vol. 36, issue 3, pp. 181-198, March 2005.

[29] K. Mori, Y. Otomo, and H. Yoshimura, "Parallel processing of 3D rigid-plastic finite element method using diagonal matrix," *Journal of Materials Processing Technology*, vol. 177, issues 1-3, pp. 63-67, July 2006.

[30] B. Bruegge and A. H. Dutiot, *Object-Oriented Software Engineering Using UML, Patterns, and Java*, Pearson Prentice Hall, New Jersey, 2004.

[31] S. R. H. Hoole and T. Arudchelvam, "Reverse engineering as a means of improving and adapting legacy finite element code," in *Proc. IEEE ICIIS-2009*, Sri Lanka, 2009, pp. 227-232.

[32] S. R. H. Hoole, T. Arudchelvam, and J. Wijayakulasooriya, "Reverse engineering legacy finite element code," *Journal of Material Science Forum*, vol. 721, pp. 307-312, 2012

[33] T. Arudchelvam, J. Wijekulasooriya, and S. R. H. Hoole, "Comparison of performance of finite element codes in different programming languages converted from legacy finite element codes," in *Proc. 3rd International Conference on Electrical, Computer, Electronics & Biomedical Engineering (ICECEBE'2013)*, Singapore, April 29-30, 2013, pp. 147-151.

[34] S. R. H. Hoole and T. Arudchelvam, "A formal UML reliant software engineering approach to finite element software development for electromagnetic field problems," *Revue Roumaine Des Sciences Techniques Serie Electrotechnique Et Energetique*; vol. 56, no. 1, pp. 5-14, 2011.

T. Arudchelvam got the B.Sc. (Hons) in Jaffna and M.Sc. in Peradeniya. He is a lecturer at Wayamba University of Sri Lanka. He has been doing research for M.Phil. He has been doing one part of this research at the University of Bath, UK and another part of the research was carried out at the Rensselaer Polytechnic Institute (RPI), USA.

S. Ratnajeevan H. Hoole got the B.Sc. Eng. Hons Cey. and M.Sc. at Mark of Distinction London. Besides, he got the Ph.D. in Carnegie Mellon. He is a professor of Electrical and Computer Engineering at Michigan State University in the US. For his accomplishments in electromagnetic product synthesis the University of London awarded him its higher doctorate, the D.Sc. (Eng.) degree in 1993, and the IEEE elevated him to the grade of Fellow in 1995. Prof. Hoole has been the vice chancellor of University of Jaffna in Sri Lanka, and as a member of the University Grants Commission there, was responsible with six others for the regulation of the administration of all 15 Sri Lankan universities and their admissions and funding. He has contributed widely to the learned literature on Tamil studies and been a regular columnist in newspapers. Prof. Hoole has been trained in Human Rights Research and Teaching at The René Cassin International Institute of Human Rights, Strasbourg, France, and has pioneered teaching human rights in the engineering curriculum.

Janaka Wijayakulasooriya got the B.Sc.Eng. and the first class honours in Peradeniya, he got the Ph.D. at Northumbria. He is a senior lecturer at University of Peradeniya. He was awarded the prof. E.O.E. Perera Gold Medal in 1994 for the most outstanding graduate of the Faculty of Engineering. He was the founder deputy chair of IEEE Sri Lanka central region subsection in 2007 and the chair of the IEEE Sri Lanka Central Region subsection in 2010.