UDC 004.925:539.3
DOI: 10.15587/1729-4061.2022.268018

DESIGNING AN OBJECT-ORIENTED ARCHITECTURE FOR THE FINITE ELEMENT SIMULATION OF STRUCTURAL ELEMENTS

Oleksii Gnezdovsky
Senior Lecturer
Department of Informational Technologies in Tourism
National University "Zaporizhzhia Polytechnic"
Zhukovskoho str., 64, Zaporizhzhia, Ukraine, 69063
Oleksii Kudin
PhD, Associate Professor
Department of Software Engineering**
Yuriy Belokon
Corresponding author
Doctor of Technical Sciences, Associate Professor*
E-mail: belokon.zp@gmail.com
Dmytro Kruglyak
PhD, Associate Professor
Department of Pressure Metal Processing**
Sergii Ilin
PhD
Department of Thermal Power and Hydropower**
*Department of Pressure Metal Processing**
**Zaporizhzhia National University
Zhukovskoho str., 66, Zaporizhzhia, Ukraine, 69600

This paper reports the development of an architecture and software implementation of the library of classes for the finite-element analysis of problems in the theory of elasticity with an open-source code. The practical necessity of such systems is due to the fact that in modern equipment there are new types of materials whose structural elements' calculation has certain features. As a result, it is necessary to update the relevant scientific software or even devise a new one. A flexible software architecture is designed to reduce the time and complexity of such updates. Existing implementations of the method of finite elements with open source have been analyzed: it was revealed that there are no systems aimed at the most flexible and user-friendly architecture. The system of abstract classes proposed in the current work corresponds to known SOLID principles of object-oriented design and makes it possible to scale the already developed analysis program for new tasks in an easy and understandable way. To test the quality of the developed system from the point of view of software engineering, the maintainability index and cyclomatic complexity code metrics were used. The values of these metrics for the modules of the PyFEM system core vary in the following ranges: from 1 to 18 for the maintainability index, and from 22 to 100 for cyclomatic complexity. PyFEM testing was performed on the task of determining the stressed-strained state of the turbine rotor blade. Due to the ease of implementation, it was possible to build a set of effective and intuitive classes that make it possible to solve numerically the static and dynamic problems in the theory of elasticity. The developed class library can be used in the development of both universal and specialized software designed to analyze multiphysics problems

Keywords: finite element method, object-oriented programming, design pattern, theory of elasticity, PyFEM

Received date 23.09.2022
Accepted date 25.11.2022
Published date 30.12.2022

***How to Cite**: Gnezdovsky, O., Kudin, O., Belokon, Y., Kruglyak, D., Ilin, S. (2022). Designing an object-oriented architecture for the finite element simulation of structural elements. Eastern-European Journal of Enterprise Technologies, 6 (2 (120)), 78–84. doi: https://doi.org/10.15587/1729-4061.2022.268018*

1. Introduction

Computer modeling of complex technical systems makes it possible to replace the study of a physical object with a computational experiment. This can significantly reduce the cost of design by reducing the number of full-scale tests. At the same time, the requirements for the computer models themselves and the corresponding software are increasing. One of the most effective and universal numerical methods for solving multiphysics problems is the method of finite elements [1]. A significant number of both universal and specialized software packages have been developed for its application. Among the most well-known programs of finite-element analysis worth noting are Abaqus, Ansys, Nastran, and many others. However, most of these programs are proprietary, that is, based on a closed program code. An alternative to them is software with an open architecture, in particular, FreeFEM, GetFEM, FreeCAD, etc. [2]. Open-source code allows the academic community to verify the software implementation of the algorithms used.

The number and variety of such systems are growing rapidly since in practice new types of materials (composites, materials with memory, etc.) are increasingly used. There are also new types of computing equipment (multiprocessor systems with shared or distributed memory, graphics and tensor processors). Thus, for the productive use of existing computer equipment and for taking into account the peculiarities of new structural materials, it is necessary to develop new software for finite-element analysis with an open-source code.

Due to the need for constant modernization of scientific software, the task arises of developing such a structure of the main modules of the programs, which would be as flexible and open to scaling as possible. As a result, so-called pat-

terns or templates for the development of scientific software systems are devised [3].

This paper proposes a methodology for developing an object-oriented architecture for the finite-element analysis of problems in the theory of elasticity, which was implemented in the Python programming language.

The relevance of the work is associated with the need to design and develop software systems for engineering analysis, with architecture adapted for further expansion, when solving new classes of problems.

2. Literature review and problem statement

Finite-element analysis systems are usually designed to solve multiphysics problems. The emergence of new materials (compositional, functional-gradient) requires constant updating of existing ones and the development of new implementations of the finite element method.

Article [4] describes a Python implementation of the processor and postprocessor, designed to solve multiphysics problems in 1D, 2D, and 3D spaces. In particular, a module for the homogenization of composite materials has been implemented. The system has an abstract interface for many grid generators. Abstract classes for methods for solving multiphysics problems (linear, nonlinear, etc.) are also implemented. But the issues of scaling this system to fundamentally different types of tasks, adding new finite elements, etc. remained uncovered. Obviously, this is due to the orientation of developers to certain classes of problems and appropriate approaches to solving them.

Work [5] considers the Freefem++ system, which implements the method of finite elements with a built-in language of a high level of description of problem statement and geometric regions. There is built-in grid generator, triangular finite elements of different types, implementation of parallelization by means of the MPI library. In general, the Freefem++ system is an integrated open-source environment in the C++ programming language using generalized programming. Such a software implementation makes it possible to achieve high speed of calculations, but further modification of the system requires highly qualified developer and is quite complex.

The general scheme of construction of a finite-element kernel for solving differential equations in a weak form is proposed in [6]. It is assumed to use symbolic calculations for the kernel of the finite element method. The software is implemented in the Julia programming language, and the use of the CUDA library and computing on graphics processors is implied. It should be noted that owing to the use of the Julia language, the system has an intuitive source code but, at the same time, is quite fast. The cited article left undisclosed the issue of software architecture and the interaction of software modules.

Work [7] describes the automated finite-element analysis system GetFEM, an essential part of which is the GWFL task description language built into the runtime library. Owing to GWFL, it is possible to set directly the mathematical formulation of the system's energy functionality. GWFL expressions are passed as text string arguments to the available functions of C++, Python, Scilab to perform calculations. The cited work does not indicate the issues of support and expansion of this system to a wider class of tasks. The possible reason is that since GetFEM is implemented in the C++ programming language, scaling the system requires considerable effort compared to Python and Julia.

Work [8] offers a framework for the development of finite-element modeling fully implemented by the Julia programming language. It is noted that the Gridap system is not a shell to other engineering analysis systems and, owing to the use of Julia, it combines a high level of abstraction of program code and the efficiency of low-level C\C++ functions. The cited work does not indicate the features of the software architecture and the ability to scale.

In addition to standard implementations using deterministic numerical algorithms, neural network variants of the finite element method have become popular in recent years.

The algorithm for constructing a neural network, which is based on the idea of sampling the method of finite elements, is described in [9]. The kernel of neural solvers of differential equations is deep neural networks, which can represent arbitrarily complex functions from the domain of values and, therefore, can approximate the equation. The system algorithm is based on the methods of Galerkin and Rayleigh-Ritz. The work does not define the issues of the speed of the system, compared with the traditional implementations of the method of finite elements. This is probably due to the fact that the system is still at the stage of development.

Paper [10] describes the FEM-NN library. The geometric domain determined by the differential equation is sampled in space using a well-established system of finite elements FEniCS [2]. The approach makes it possible to train neural networks when solving the optimization problem where the equation itself and boundary conditions are constraints. The work does not define the limits of application of the software implementation and the software repository does not contain sufficient documentation.

The authors of work [11] propose a dynamic architecture of deep learning to solve linear parametric partial differential equations. The connections between neurons in architecture mimic the coherence graph of finite elements during the application of grid refinements. The software implementation is made in Python. The article does not highlight the features of the software implementation, which is important from the point of view of further practical use of the system.

One of the important stages of the work of approximate methods for solving boundary problems is the solution of systems of linear algebraic equations. Work [12] proposes a method for solving large branched systems of algebraic equations. This approach involves the representation of a system of equations in the form of an unoriented weighted graph, which is the input of a graph neural network. The neural network is trained to solve the problem of regression of the system solution. Testing on static linear problems of structural mechanics has been carried out. It should be noted that the method is less accurate compared to classical numerical methods, and slower at that. However, as shown in the article on a synthetic example, neural network approximations may be promising.

In [13], the application of such neural networks to the solution of partial differential equations is analyzed: networks based on radial-basis functions (RBF-networks), multilayer perceptrons, and cell networks. Theoretical calculations are given, without describing the software implementation, which is a disadvantage, from the point of view of a given article.

So, from the above works on the development of systems of finite-element analysis, we can draw the following conclusions:

– when implementing classical variants of the finite element method, often the computational core consists of modules in C++ to speed up operations [5]. At the same time, further scaling and adding new modules is problematic;

– recently, there have been wider distributed software implementations in the Julia language [6, 8]. However, an analysis of the source code showed that scaling these systems is problematic for a third-party user;

– a common approach is the implementation of neural network variants of approximate methods while the neural network is considered as a universal approximator. Currently, this approach does not always give an advantage in accuracy or performance [12]. However, neural network numerical methods are promising and require further research;

– there are no works with a focus on applying the principles of object-oriented design to the development of methods for constructing scalable systems of finite-element analysis.

Among the above implementations of the finite element method, there are no advancements aimed at potentially expanding the capabilities of the system by its users. Most studies consider integral systems focused on a specific type of analysis, and changes to such systems can be carried out either by developers or qualified programmers.

3. The aim and objectives of the study

The aim of this study is to devise a methodology for designing object-oriented systems of finite-element modeling, which can be used to build both universal and specialized software designed to analyze multiphysics problems.

This will make it possible not only to use the developed software as a finished product but also to scale it in an easy and understandable way.

To accomplish the aim, the following tasks have been set:

– to design an open object-oriented architecture of the system of finite-element analysis;

– to develop an object-oriented PyFEM class library, which implements the solution of static and dynamic problems of the theory of elasticity using the method of finite elements;

– to test the developed system on model problems from the theory of elasticity.

4. The study materials and methods

The object of our study is the systems of finite-element analysis. The focus is on open software implementations in the form of a package of modules or independent software. The hypothesis of the study assumes that software implementations of the finite-element method with a scalable architecture will improve the quality of research using finite-element analysis. It is assumed that the object-oriented programming paradigm is the most acceptable for the development of this class of systems. Accepted simplifications are that a qualitative analysis of the program code of the systems considered in the work and relevant publications is used. The main criteria are the presence of a clear class structure that corresponds to the main stages of the finite element method.

Software complexes of finite-element analysis, as a rule, consist of three basic subsystems:

1) preprocessor – subsystems of automation of preparation of initial data for calculation (generator of finite-element model of the original geometric domain);

2) processor – the core of the analysis system, which directly implements the finite-element calculation of the problem (statics, dynamics, thermoelasticity, etc.);

3) postprocessor – a subsystem that automates the analysis of the obtained numerical results, for example, by their specific visualization [1].

As a preprocessor, the work can use, for example, Gmsh [14], Netgen/NGSolve [15].

The main component of any finite-element system is the processor. It directly implements one or another algorithm for applying the method of finite elements to solve a specific class of problems. Usually, this algorithm makes it possible to construct for each finite element its corresponding matrices of hardness, mass, and damping, which depend on the energy functionality used (variational principle). All further functions of the processor (or solver) are fairly standard. Among them are adding local matrices of finite elements to their corresponding global ones; taking into account the boundary conditions; solving systems of linear algebraic equations, etc. The architecture of the processor in general depends on the type of tasks on which it is focused (for example, statics or dynamics) but in most cases, it is closed (that is, it does not allow its easy expansion to adapt to new conditions of use) [1].

Since the result of the work of any processor of the finite-element analysis system is a large array of numerical information, the task arises of increasing the clarity of its analysis. For this purpose, so-called postprocessors are being developed that automate the analysis of numerical information. Usually, visualization is used for this in the form of various graphs or color images of the distribution of the values of the resulting function – a phase variable in the calculation area. Also, prostprocessors can synthesize additional information (for example, calculating the intensity of stresses based on the values of previously obtained components of the stress tensor). The substantiation of the above structure of finite-element analysis systems is given in [1].

Software development and support are becoming increasingly complex and time-consuming processes. One of the main approaches to the design of complex software systems is the object-oriented programming paradigm [16]. When using this paradigm, in particular for the development of scientific software, certain architectural templates are typically used. These may include some generic solutions in software architecture that are applied in a specific given context. More general approaches to the design of object-oriented systems are also used, for example, SOLID principles that describe the general requirements for class construction and interaction between them [16].

Among the main generally accepted requirements for the construction of object-oriented software are the following: separation of class responsibility according to the functions of the developed system; openness of classes for expansion; avoiding duplication of information and program code in the system; the simpler the class structure, the better the rest [1, 16]. The use of these principles in the design will make it possible to develop a software implementation open for adding new implementations of the method of finite elements, types of elements, methods of preprocessor and postprocessor processing, etc. That is, it will make the program open to scaling and expanding functionality.

So, when developing complex software systems, in particular finite-element analysis programs, it is relevant to apply an object-oriented paradigm and general design principles, for example, SOLID.

5. Design and development of a finite-element analysis system

5. 1. Design of an open object-oriented architecture of a finite-element analysis system

The process of designing the architecture of the system of finite-element analysis is partially described in [17]; this article expands the description of the main abstract classes and model problems.

The description of the discrete model of the original domain can be given in the form of a class hierarchy, where the basic abstract class, TMesh (Fig. 1), contains all the information necessary for calculation about the structure of the grid: the type of finite elements; the number of nodes, etc., but does not have the implementation of downloading data from a file of a specific format (vol, mesh, trpa, etc.). Classes derived from TMesh (for example, TMeshTRPA) contain only methods for reading information from grid data files in a given format [17].

The most important from the point of view of software implementation of the finite element method are classes that encapsulate different types of finite elements (their elastic and physical characteristics, local stiffness, mass and damping matrices, etc.). To account for their diversity, PyFEM implements a hierarchical class structure. The base is the abstract class TFE, which describes the most fundamental properties of an isoparametric finite element: the number of nodes (dimension); cross-sectional area for one-dimensional or thickness for two-dimensional elements; elastic properties; temperature; coefficient of thermal expansion; density; damping factor; local hardness, mass and damping matrices, quadrature parameters for numerical integration, etc. In this class, the procedure for constructing local matrices is defined but not implemented because it depends on the type of a particular element. Abstract classes TFE1D, TFE2D, and TFE3D derived from TFE implement the construction of local matrices for standard one-, two-, and three-dimensional elements [17]. These implementations do not contain appropriate procedures for constructing functions of finite element forms (Fig. 1).

No less important component of any system of finite-element analysis is the module, which directly implements the algorithm for solving a specific type of problem [1]. Due to a large number of types of calculations (statics, dynamics, nonlinearity, contact problems, etc.), their universal practical implementation is associated with certain difficulties. The PyFEM library has developed a number of interrelated classes for this, each of which is designed to solve a specific type of problem.

The basic abstract class TFEM contains the most general properties and methods necessary for the programmatic implementation of the finite element method: a description of the finite-element grid; all the parameters necessary for the calculation; solver of systems of linear algebraic equations; table of results, etc. The central method of TFEM is the procedure for starting the calculation, which in this class is abstract since its specific implementation depends on the type of task. PyFEM currently implements two successor classes from TFEM: TFEMStatic and TFEMDynamic, designed to solve the corresponding elastic static and dynamic problems (Fig. 2) [17].

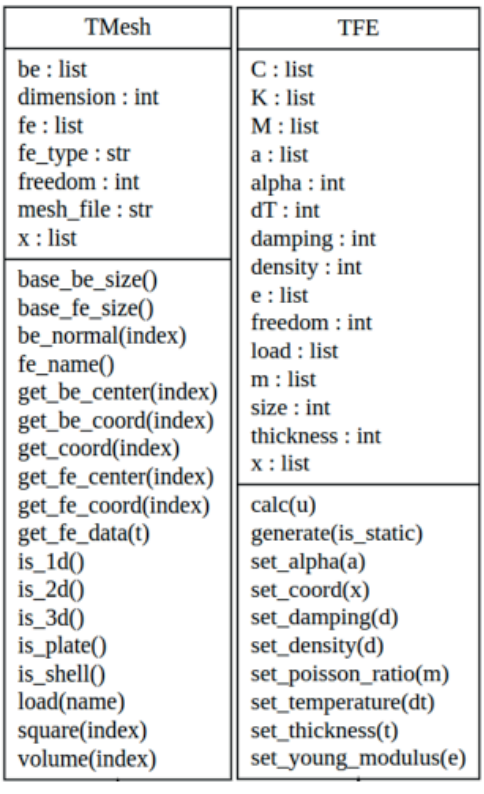


Fig. 1. UML diagram of the main abstract classes of the discrete model (TMesh) and finite element (TFE)

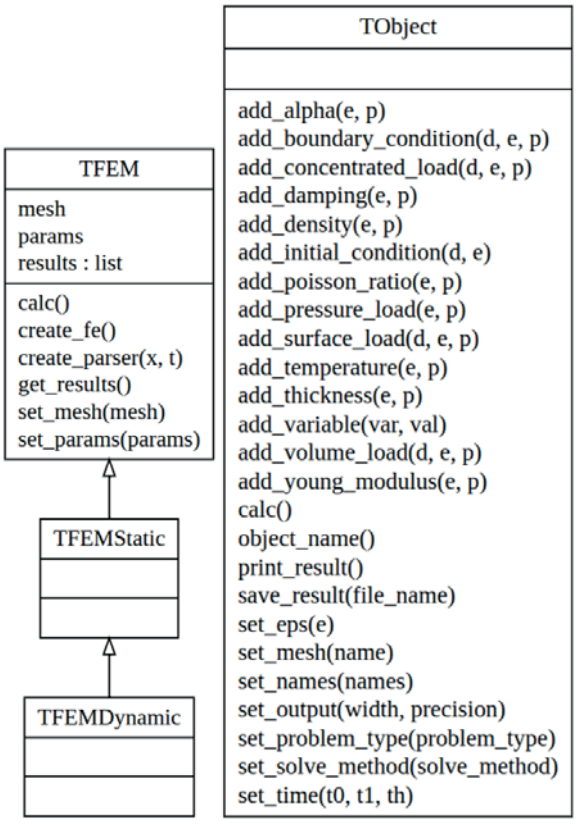


Fig. 2. UML diagram of abstract classes of the finite element method (TFEM) and calculation object (TObject)

The central part of the PyFEM library is the TObject class (Fig. 2). In fact, it is a shell for TFEM and its heirs and implements a user interface for accessing the library. By controlling the methods of this class, the user can determine a new object of calculation, and assign a finite-element model to it, elastic and physical characteristics, boundary conditions and loads, as well as other parameters necessary for calculation. The user can also choose a way to solve systems of linear algebraic equations (direct or iterative), the format for displaying the calculation results in a file or screen, etc. [17].

5. 2. Software implementation of the PyFEM class library

The software implementation of the finite element method class library is performed in the Python programming language using the basic libraries Math, Scipy, Numpy. So, the work of PyFEM does not require the installation of additional libraries in which version conflicts may occur.

Fig. 3 shows an example of creating an instance of the TObject class for a new calculation. In fact, to perform finite-element analysis using the PyFEM library, it is necessary to connect all the necessary libraries and create a TObject object with specified physical and geometric parameters.

In the above example (Fig. 3), by using the method of class TObject set_mesh(), from a given data file one loads information about the discrete shell model that was built in the Gmsh application [14]. After that, the calculation type (method set_problem_type()), calculation method (set_solve_method() method), Young's module and Poisson coefficient (methods add_young_modulus() and add_poisson_ratio(), respectively), boundary conditions (method add_boundary_condition()) are set, and the volumetric load (add_volume_load() method) is determined.

To determine the effectiveness of the software implementation, code metrics are used. Among the most popular metrics are the following: maintainability index (English: Maintainability Index, MI), cyclomatic complexity (English: Cyclomatic Complexity, CC) [18].

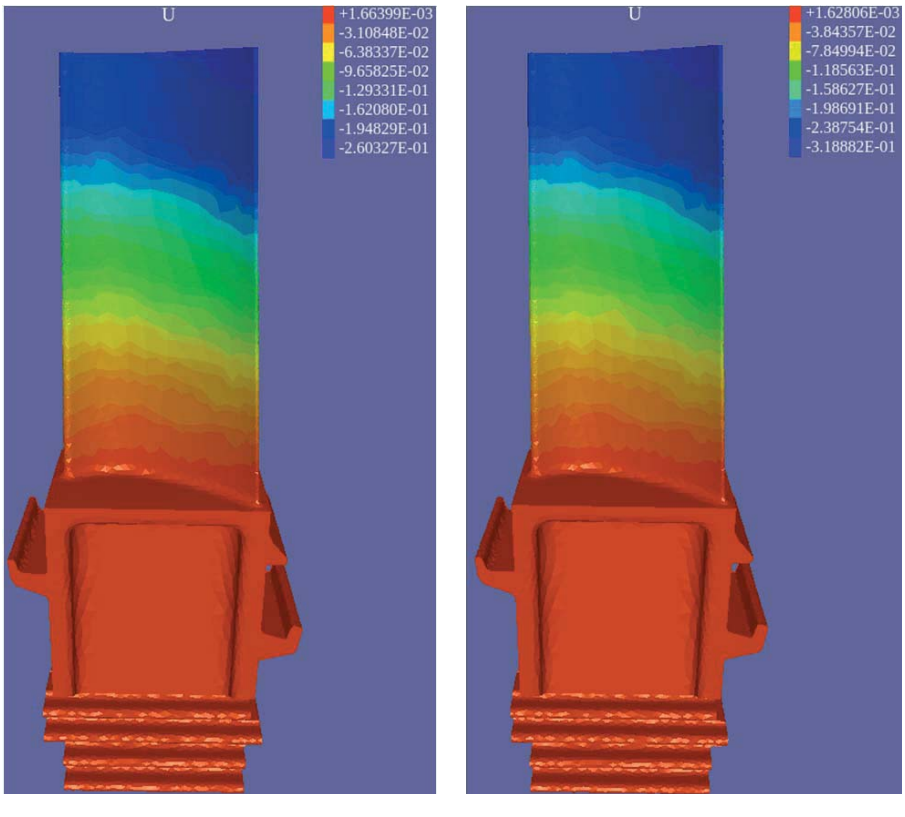
As shown in [18], these dimensionless numerical characteristics determine the quality of the program code, in terms of its support and scaling. The MI value in the range of 20–100 indicates a satisfactory level of complexity of code support. The SS metric indicates the structural complexity of the code; at values greater than 10, it is recommended to refactor [18]. The values of these metrics for the modules of the PyFEM system core vary in the following ranges: MI from 1 to 18, SS from 22 to 100. In general, this indicates a satisfactory quality of the program code in terms of further scaling. To calculate the metrics, the Radon package was used (https://pypi.org/project/radon/).

```
def blade(res_name):
    obj = TObject()
    if obj.set_mesh('mesh/blade.trpa'):
        obj.set_problem_type('static')
        obj.set_solve_method('direct')
        obj.add_young_modulus('95000')
        obj.add_poisson_ratio('0.3')
        obj.add_boundary_condition(DIR_X | DIR_Y | DIR_Z, '0', 'z <0')
        obj.add_volume_load(DIR_Y | DIR_X, '-0.05', '')
        if obj.calc():
            obj.print_result('mesh/' + obj.object_name() + '.res')
            obj.save_result(res_name)
            TPlot(res_name)
            return True
        return False
```

Fig. 3. Software implementation of the calculation of the turbine rotor blade

5. 3. Testing on model problems from the theory of elasticity

As a numerical example, the problem of determining the stressed-strained state of the turbine rotor blade is considered (Fig. 4). The following materials were used to compare the results: Titanium Ti-6Al-4V and an intermetallic alloy based on titanium aluminides Ti-Al-Nb. The physical properties of these materials are given in Table 1.



a *b*

Fig. 4. Turbine rotor blade calculation results: *a* — Ti-6Al-4V material; *b* — Ti-Al-Nb material

Table 1

Physical properties of materials for a numerical experiment

Material	Young's modulus, MPa	Poisson coefficient
Ti-6Al-4V [19]	1.14E05	0.342
Ti-Al-Nb [20, 21]	0.95E05	0.300

The calculations were carried out in a linear-elastic statement with a volumetric load of 0.05 MPa, the type of finite element was a linear tetrahedron, 4 nodes. There are no temperature stresses.

Fig. 4 shows the distribution of movements U along the X axis thru the volume of the blades.

The software implementation of the developed PyFEM class library can be found at the https://github.com/SeregaGomen/pyfem link.

6. Discussion of results of the design and development of an object-oriented system

The developed project of the system of finite-element analysis and the open library of classes as a whole correspond to the approaches to object-oriented programming, in particular, SOLID principles [16]. The PyFEM system design methodology, a system of developed classes, is common for the implementation of the finite element method and can be used in the development of other similar software systems.

The developed architecture makes it possible to add new elements of the system by imitating abstract classes. For example, an abstract class of a finite element TFE (Fig. 1) must be the parent for any implementation of a new element type. Similarly, when creating a new version of the finite element method, for example, for the thermoelasticity problem, it is necessary to imitate the abstract class TFEM (Fig. 2) and redefine the methods of this class.

In general, this approach to development is flexible and makes it possible to use PyFEM in the development of other engineering analysis systems or directly for calculations.

The software implementation of PyFEM is fully executed by the Python programming language (Fig. 3), while, for programming the processor modules, the Math, Scipy, Numpy libraries are used, which have stable versions for different operating systems. In addition, the high level of abstraction of the Python language can reduce development time, compared to the C++ or Fortran languages. Analysis of similar implementations showed that, for example, the systems SfePy [4], FreeFem [5], GetFem [7] are implementations of the finite element method for solving a wide range of problems from the theory of elasticity, however, there is currently no possibility for quickly adding new modules to expand the possibilities of calculation. To test the quality of the developed system from the point of view of software engineering, the maintainability index and cyclomatic complexity code metrics were used. The values of these metrics for the modules of the PyFEM system core vary in the following ranges: from 1 to 18 for the maintainability index, and from 22 to 100 for cyclomatic complexity.

Model problems (Fig. 4) illustrate the work of the postprocessor to visualize the distribution of the desired movements by the volume of the structure.

Currently, the limitations of the developed system are the inability to parallel calculation using graphics processors.

The disadvantage of the study is the lack of implementation of methods for solving problems of physically and geometrically nonlinear elasticity, plasticity, viscoelasticity.

So, further development may consist in programming computational modules using computing libraries on graphics processors (for example, CUDA). As well as adding new variants of the finite element method for a wider range of tasks.

7. Conclusions

1. An open object-oriented architecture system of finite-element analysis has been designed. The hierarchical structure of classes encapsulates the object of calculation, the static and dynamic implementation of the finite element method, finite elements of different types, the discrete model of the original object, etc. The method of designing the system used is common for the implementation of the finite element method and can be applied in the development of other similar implementations of the finite element method.

2. An object-oriented PyFEM class library has been developed. Due to the ease of implementation, it was possible to build a set of effective and intuitive classes that make it possible to perform numerical solutions to static and dynamic problems from the theory of elasticity. The software system provides the ability to easily expand it to improve the functionality of the processor, which is confirmed by the calculation of source code metrics. To test the quality of the developed system from the point of view of software engineering, the maintainability index and cyclomatic complexity code metrics were used. The values of these metrics for the modules of the PyFEM system core vary in the following ranges: from 1 to 18 for the maintainability index, and from 22 to 100 for cyclomatic complexity.

3. PyFEM testing was performed on the task of determining the stressed-strained state of the turbine rotor blade. Examples of the software interface for determining the calculation conditions and the results of the postprocessor are given. Our technological advancement can be applied in the practice of design organizations.

Conflicts of interest

The authors declare that they have no conflicts of interest in relation to the current study, including financial, personal, authorship, or any other, that could affect the study and the results reported in this paper.

Acknowledgments

The study was carried out within the framework of research work 0122U001765 "Thermochemical pressing of special-purpose materials".

References

1. Breslavskyi, D. V., Korytko, Yu. M., Tatarinova, O. A. (2017). Proektuvannia ta rozrobka skinchennoelementnoho prohramnoho zabezpechennia. Kharkiv, 232. Available at: http://library.kpi.kharkov.ua/files/new_postupleniya/prropz.pdf

2. Logg, A., Mardal, K.-A., Wells, G. (Eds.) (2012). Automated Solution of Differential Equations by the Finite Element Method. Lecture Notes in Computational Science and Engineering. doi: https://doi.org/10.1007/978-3-642-23099-8
3. Choporov, S., Gomenyuk, S., Kudin, O., Lisnyak, A. (2018). Design patterns for object-oriented scientific software. CEUR Workshop Proceedings, 441–444. Available at: https://ceur-ws.org/Vol-2105/10000441.pdf
4. Cimrman, R., Lukeš, V., Rohan, E. (2019). Multiscale finite element calculations in Python using SfePy. Advances in Computational Mathematics, 45 (4), 1897–1921. doi: https://doi.org/10.1007/s10444-019-09666-0
5. Hecht, F. (2012). New development in freefem++. Journal of Numerical Mathematics, 20 (3-4). doi: https://doi.org/10.1515/jnum-2012-0013
6. Xie, J., Ehmann, K., Cao, J. (2022). MetaFEM: A generic FEM solver by meta-expressions. Computer Methods in Applied Mechanics and Engineering, 394, 114907. doi: https://doi.org/10.1016/j.cma.2022.114907
7. Renard, Y., Poulios, K. (2021). GetFEM: Automated FE Modeling of Multiphysics Problems Based on a Generic Weak Form Language. ACM Transactions on Mathematical Software, 47 (1), 1–31. doi: https://doi.org/10.1145/3412849
8. Badia, S., Verdugo, F. (2020). Gridap: An extensible Finite Element toolbox in Julia. Journal of Open Source Software, 5 (52), 2520. doi: https://doi.org/10.21105/joss.02520
9. Khara, B., Balu, A., Joshi, A., Sarkar, S., Hegde, C., Krishnamurthy, A., Ganapathysubramanian, B. (2021). NeuFENet: Neural Finite Element Solutions with Theoretical Bounds for Parametric PDEs. arXiv. doi: https://doi.org/10.48550/arXiv.2110.01601
10. Mitusch, S. K., Funke, S. W., Kuchta, M. (2021). Hybrid FEM-NN models: Combining artificial neural networks with the finite element method. Journal of Computational Physics, 446, 110651. doi: https://doi.org/10.1016/j.jcp.2021.110651
11. Uriarte, C., Pardo, D., Omella, Á. J. (2022). A Finite Element based Deep Learning solver for parametric PDEs. Computer Methods in Applied Mechanics and Engineering, 391, 114562. doi: https://doi.org/10.1016/j.cma.2021.114562
12. Grementieri, L., Galeone, P. (2022). Towards Neural Sparse Linear Solvers. arXiv. doi: https://doi.org/10.48550/arXiv.2203.06944.
13. Trushevskyi, V. M., Shynkarenko, H. A., Shcherbyna, N. M. (2014). Metod skinchennykh elementiv i shtuchni neironni merezhi. Liviv: LNU imeni Ivana Frankach, 396.
14. Geuzaine, C., Remacle, J.-F. (2009). Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities. International Journal for Numerical Methods in Engineering, 79 (11), 1309–1331. doi: https://doi.org/10.1002/nme.2579
15. Netgen/NGSolve. Available at: https://ngsolve.org/
16. Weisfeld, M. (2019). The Object-Oriented Thought Process. Addison-Wesley, 412.
17. Ihnatchenko, M. S., Kudin, O. V., Gnezdovskiy, O. V. (2020). Object-oriented implementation of the finite element analysis library in the python programming language. Visnyk of Zaporizhzhya National University. Physical and Mathematical Sciences, 1, 138–147. doi: https://doi.org/10.26661/2413-6549-2020-1-18
18. Turan, O., Tanriöver, Ö. Ö. (2018). An Experimental Evaluation of the Effect of SOLID Principles to Microsoft VS Code Metrics. AJIT-e: Online Academic Journal of Information Technology, 9 (34), 7–24. doi: https://doi.org/10.5824/1309-1581.2018.4.001.x
19. Ranjan, A., Rakshith, A. (2021). Analysis of Industrial Gas Turbine Blade. International Research Journal of Engineering and Technology, 8 (5), 4247–4251.
20. Yuriy, B., Aleksandr, Z., Karina, B. (2017). The investigation of nanostructure formation in intermetallic γ-TiAl alloys. 2017 IEEE International Young Scientists Forum on Applied Physics and Engineering (YSF). doi: https://doi.org/10.1109/ysf.2017.8126640
21. Sereda, B., Sereda, D., Belokon, Y. (2015). Investigation of corrosion and oxidation of γ-TiAl alloys obtained in self propagating high temperature synthesis. Materials Science and Technology Conference and Exhibition. Vol. 2. Columbus, 1249–1255.