Scientific Journal of the Ternopil National Technical University
2026, vol. 122, no. 2, pp. 84–94,
ISSN: 2522-4433 Print, ISSN: 2522-4441 Online
https://doi.org/10.35784/ visnyk_tntu2026.02.084

Submitted: 2026-02-21 | Revised: 2026-05-18 | Accepted: 2026-05-26

An extensible software architecture for modeling complex object-oriented multicomponent systems

Mykhaylo Petryk[1], *, Dmytro Mykhalyk[2], Mykola Zaiarnyi[3], Oleksiy Tsebriy[4], Mykola Shynkaryk [5]

[1]Ternopil Ivan Puluj National Technical University, Ukraine, mykhaylo_petryk@tntu.edu.ua

[2]Ternopil Ivan Puluj National Technical University, Ukraine, d.mykhalyk@gmail.com

[3]Ternopil Ivan Puluj National Technical University, Ukraine, mykola_zaiarnyi1707@tntu.edu.ua

[4]Ternopil Ivan Puluj National Technical University, Ukraine, ol.tsebriy@gmail.com

[5] West Ukrainian National University, Ukraine, shynkaryk_m@ukr.net

*Corresponding author mykhaylo_petryk@tntu.edu.ua

Abstract: *This article presents a modular, object-oriented architecture for the PBM (Porous Media Modeling) library, a Java-based system designed to model consolidation filtering kinetics in hierarchical porous media with two- and three-level porosity structures. Developing extensible software architectures for scientific computing remains a persistent challenge, particularly for complex, multi-component systems with multi-level relationships. Current software modeling tools are tightly coupled with specific object-oriented domains and the mathematical models of physical processes, such as consolidation filtering in porous media. Consequently, these tools must satisfy strict engineering requirements regarding architectural stability, feature expansion, and the seamless integration of new numerical schemes and physical models. The proposed architecture consists of four decoupled modules: Process, Direct, GUI, and Extra. These modules are interconnected through well-defined interfaces and governed by the Factory, Strategy, and Template Method design patterns. Specifically, the Process, Direct, and Extra modules encapsulate simulation parameters and implement diverse methods for computer modeling and parameter identification. These components control the spatiotemporal distributions of key process variables across various macro- and micro-porous scales. Concurrently, the GUI module provides polymorphic visualization via the drawable interface. Ultimately, we demonstrate that this architectural design enables the seamless integration of new mathematical models, numerical schemes for high-performance computing, and analytical methods without modifying the existing codebase, thereby satisfying the open-closed principle.*

Keywords: *software architecture; object-oriented design and programming, mathematical software modeling, UML, extensible architectural models, design patterns, software requirements, use cases, complex multi-component and muliyporous systems, computational procedure scenarios.*

1. INTRODUCTION

Advanced software architecture methods are essential for modern research platforms. These platforms focus on mathematical and computer modeling of complex, multi-component object-oriented systems and processes. They use science-intensive models to simulate the transfer and transformation of mass, energy, electric charge, filtrate, and momentum [1–3].

The qualitative results of researches have wide applications in healthcare, medicine, ecology, and chemical engineering. In particular, they help address environmental pollution and manage toxic waste landfills. These biochemical wastes are often in complex

transitional states, posing severe threats to groundwater basins, soil mechanics, and materials science [4–6].

Mathematical models of these processes use systems of interconnected partial differential equations (PDEs). Based on fundamental laws and equilibrium conditions, these equations describe substance transformation and parameter redistribution across micro- and macro-processes. Specifically, they model time-space pressure and concentration distributions within inter-particle macropores and intraparticle spaces [7]. Solving these systems requires efficient numerical methods for high-performance computing scenarios. These scenarios rely on implicit computational schemes, convergent Crank-Nicolson algorithms, finite difference methods, and fast, low-cost analytical solutions [8]. These physical models are becoming increasingly complex. Researchers must constantly update software requirements to include new kinetic factors, such as multi-level and graph-like porosity, non-linear consolidation functions, and phase transformations in various micro-particle geometries. This growing complexity creates a strong demand for software frameworks that can handle diverse mathematical formulations. Such frameworks must support changing research requirements without requiring developers to constantly rewrite the core program code.

Several existing scientific computing frameworks can partially address this problem. For example, FEniCS [9] provides a general finite-element environment with diverse solvers based on an object-oriented factory pattern. OpenFOAM [10] uses a modular architecture to separate solvers, meshes, and post-processors. Similarly, MATLAB toolkits [11] decouple computational cores from visualization components. However, these frameworks are too general and lack the necessary level of detail. Consequently, they/cannot account for the key factors required to accurately describe and analyze complex, multi-hierarchical consolidation models. In particular, they fail to model substances changing between liquid and solid states within complex, interconnected heterogeneous spaces.

Despite the recognized importance of extensible software architecture in scientific computing, there is a lack of documented, template-driven solutions for hierarchical modeling of multiscale consolidation filtering. Most existing software in this field provides either a single fixed numerical scheme or a set of custom scripts. These solutions lack an extensible architecture to support incremental updates [11, 12]. Existing research has not yet systematically analyzed how module decomposition, interface design, and design patterns can be combined to create extensible architectures for this specific domain.

2. METHODS

2.1. Study design

The main objective of this study is to design and validate a stable, extensible software architecture for modeling complex physical systems. This architecture ensures high scalability and allows adding new features without major code refactoring or breaking existing dependencies. To achieve this, we utilize UML tools and a functional requirements model. The corresponding use cases are defined by the specific needs of simulating complex physical object-oriented consolidation systems in multidimensional microporous spaces (CSMPS).

The article represents a case study in the field of research software engineering, based on the retrospective-prospective architectural analysis of the PBM library. The authors developed this mathematical framework during a long-term international scientific

cooperation with the TIMR and LMAC laboratories of the Compiègne University of Technology (France).

The study considers the source code, module structure, class hierarchies and intermodule dependencies to define and document the architectural patterns that control the system. The analysis supplemented with new promising system tools and functional design elements that significantly expand the architectural possibilities of supplementing the system software with new functionalities.

2.2. Analysis of research software engineering requirements of CSMPS

Scientific requirements for the physical essence of the studied CSMPS and methodologies for their modeling and research were described in [8, 13, 14]. These researches based on the analysis of the latest world approaches to the study of such systems and objects according to works [2–7]. As a result, we defined an iterative, multi-step software development process driven by scientific requirements. This process based on both a use-case model and an architecture-centric design.

2.3. Approach to architectural analysis

Architectural analysis carried out in three stages. First, the module boundaries determined by examining Java package declarations and import statements in all source files. Second, the class hierarchies in each module reconstructed from inheritance declarations and interface implementations. Third, internal module dependencies mapped by identifying which classes import types from other modules, establishing a unidirectional dependency graph [15–17].

A few identified design patterns based on the catalog definitions by Gamma et al. [18–21]: Factory patterns identified as static factory methods that return interface types; Strategy patterns defined as interface implementations passed as parameters; Template Method patterns identified as abstract classes that define algorithmic skeletons with abstract steps; Composition patterns (or composition principles) were defined as classes that contain references to other objects rather than inheriting from them [22–24].

All equations of the mathematical model of the CSMPS governing the simulated and studied physical processes formulated in [13] reviewed in the context of the main characteristics of the domain requirements model, from the point of view of their impact on architectural solutions.

2.4. Implementation Environment

The PBM library is implemented in Java SE using the NetBeans IDE with an Ant build system. Its GUI components are based on AWT and Swing. The library does not rely on external scientific computing packages; instead, all numerical schemes are implemented directly within the codebase. The source code is organized into packages that correspond to the four identified modules.

3. RESULTS

3.1. Overall System Architecture

The architectural analysis identified four primary modules with a strictly unidirectional dependency structure, shown in Fig. 1.

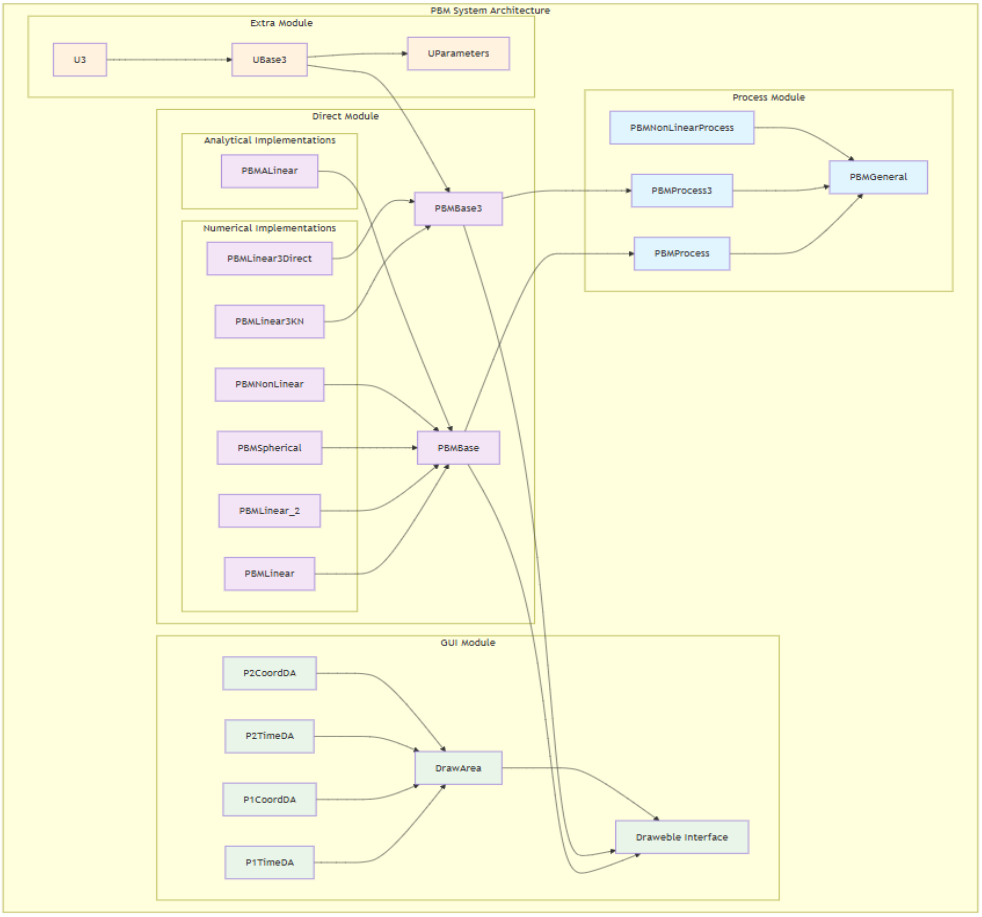

Figure 1. High-level architecture of the PBM system showing four modules and their unidirectional dependencies.

Developing extensible software architectures for scientific computing remains a persistent challenge, especially when modeling complex, multi-component systems with multi-level relationships. Existing software modeling tools are tightly coupled with the characteristics of object-oriented domains and the mathematical modeling of physical processes, such as consolidation filtering in porous media. Consequently, these tools must satisfy strict requirements regarding stability, feature expansion, and the integration of new numerical schemes and physical models.

Current research shows modular object-oriented architecture for the Porous Media Modeling (PBM) library. This Java-core system dedicated for filtration consolidation kinetics modeling in hierarchical porous media, which are characterized by two- and three-level porosity structures.

The architecture consist of four isolated modules: Process, Direct, GUI and Extra. Each module interconnected through defined interfaces and controlled by the software design patterns: Factory, Strategy and Template Method. Module Process, Direct and Extra encapsulate various modeling parameters and implement different numerical modeling parameters identification methods. The GUI module provides polymorphic visualization through the Draweble interface. The article demonstrates the capabilities of the proposed architectural model and the design of its components to integrate and develop new mathematical models, numerical schemes for high-performance computing, classes and analysis methods without changing the existing code,

satisfying the open-closed principle based on polymorphism and architectural patterns. The extensibility mechanism is illustrated by step-by-step integration procedures for new models, parameter sets and visualization components.

3.2. Direct Module

The Direct module contains all numerical and analytical solvers. Its architecture, shown in Fig. 3, uses two abstract base classes `PBMBase` for two-level models and `PBMBase3` for three-level models both implementing the `Draweble` interface.

The `PBMBase` use static factory method instantiates the appropriate solver at runtime. Each concrete subclass implements four abstract methods: `P1(time, Z)` returning macro-pore concentration, `P2 (time, X, Z)` returning micro-pore level-1 concentration, `calculation()` triggering the full numerical solve, and `initialization()` called on parameter change. Six two-level implementations are present (Table 1) and two three-level implementations.

Table 1. Implementations in the Direct module.

Class	Level	Method	Geometry
PBMALinear	2	Analytical	Planar
PBMLinear	2	Crank–Nicolson	Planar
PBMLinear_2	2	Explicit	Planar
PBMSpherical	2	Crank–Nicolson	Spherical
PBMNonLinear	2	Crank–Nicolson	Planar, nonlinear
PBMNonLinear_2	2	Explicit	Planar, nonlinear
PBMLinear3KN	3	Crank–Nicolson	Planar
PBMLinear3Direct	3	Explicit	Planar

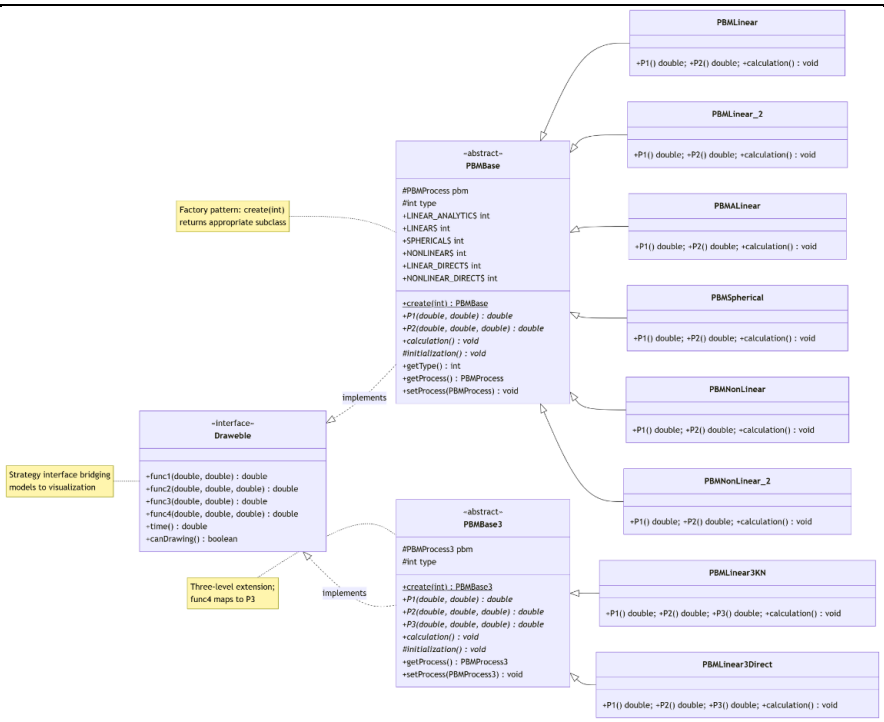

Figure 2. Class hierarchy and interface relationships in the Direct module.

3.4. GUI Module

The GUI module provides a component hierarchy for visualizing concentration profiles, governed by the `Draweble` interface as the Strategy pattern. The module structure is shown in Fig. 3.

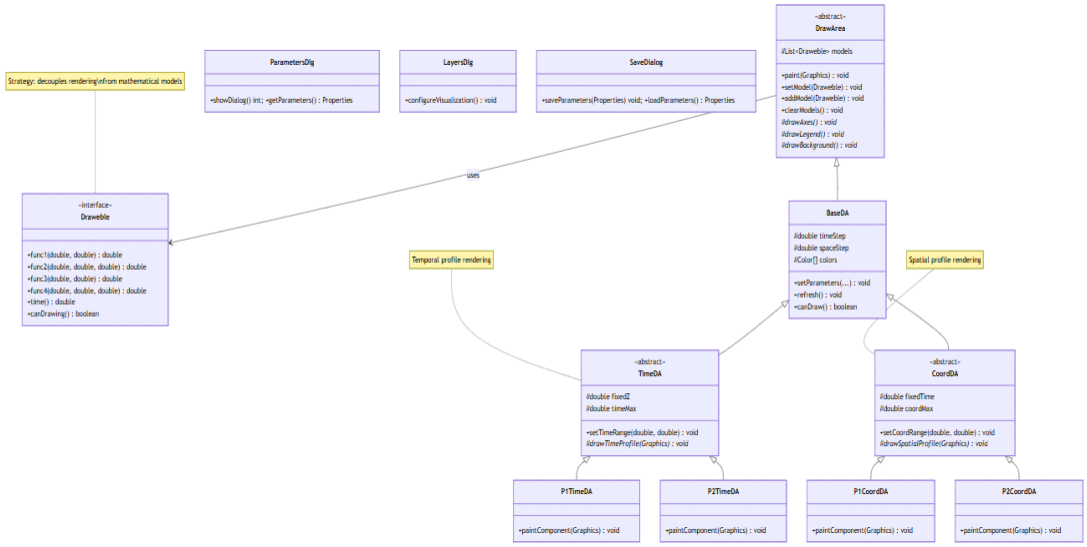

Figure 3. GUI module class hierarchy and Strategy pattern implementation.

The `Draweble` interface exposes four function slots that map to the physical quantities of interest. The `canDrawing()` method acts as a guard: rendering components call it before accessing data, preventing visualization of uncommitted or incomplete computation results.

3.5. Extra Module

The Extra module implements engineering consolidation analysis as a layered extension over the Direct module. Its class structure is shown in Fig. 4.

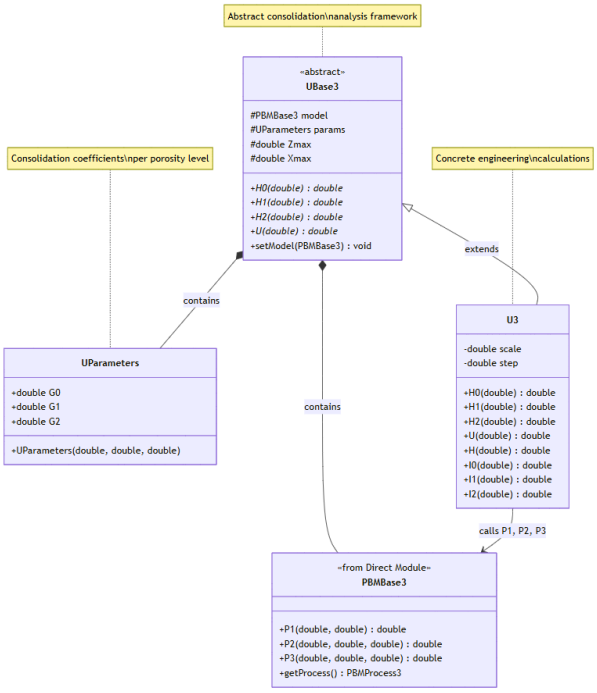

Figure 4. Extra module class structure showing consolidation analysis framework.

The abstract class `UBase3` defines the consolidation analysis interface: `H0(t)`, `H1(t)`, `H2(t)` return the settlement contributions from macro-pores and the two micro-pore levels respectively, while `U(t)` returns the normalized degree of consolidation. The concrete class `U3` implements these through spatial numerical integration of the pressure fields.

The `U3` class accesses process parameters exclusively through the model interface, maintaining loose coupling between the analysis layer and the parameter management layer.

3.6. Extensibility Mechanism

The architecture provides a well-defined three-step procedure for integrating a new two-level mathematical model without modifying any existing class. After implementing new solver by extending `PBMBase` class, registering the type constant and update the factory, the new model is immediately compatible with all existing GUI visualization components through the `Draweble` interface and can be passed to `UBase3` subclasses for consolidation analysis and no changes to other modules are required.

4. DISCUSSIONS

4.1. Design Pattern Effectiveness

The three primary patterns identified: Factory, Strategy, and Template Method, each address a distinct extensibility concern. The Factory pattern localizes instantiation logic, ensuring client code isn't updated when new models are added, only the factory method. In contrast to generic factory implementations, PBM's factory encodes domain knowledge: constants maps directly to numerical scheme categories, enabling informed solver selection based on stability requirements [19].

The Strategy pattern, implemented through `Draweble`, achieves the strictest possible decoupling between computation and visualization: a visualization component has no compile-time dependency on any concrete model class. This satisfies the Dependency Inversion Principle [19, 20]. The Template Method pattern in `PBMBase` and `PBMBase3` defines the algorithmic skeleton (`setProcess` → `initialization` → `calculation` → access `P1`/`P2`/`P3`) while delegating the computationally variable steps to subclasses, preventing incorrect usage sequences.

4.2. Comparison with Related Frameworks

Compared to FEniCS [9], PBM is domain-specific but requires no external dependencies and operates within standard Java SE, lowering the barrier to deployment in engineering practice. Compared to OpenFOAM [10], PBM's module boundaries are enforced at the Java package level rather than through a build system, providing less runtime configurability but stronger compile-time type checking. Compared to MATLAB toolboxes [11], PBM's object-oriented structure enables polymorphic visualization that MATLAB's functional programming style makes cumbersome.

4.3. Identified Limitations

Several architectural limitations were identified. First, the factory method requires modification when a new model type is added, which violates the strict Open-Closed Principle; this could be addressed by registering model providers at startup through a map-based factory or a dependency injection framework [15]. Second, public field access in `PBMProcess` and `UParameters` bypasses encapsulation, making validation impossible without breaking existing callers. Third, `Draweble` uses the name «Draweble» rather than the standard English

«Drawable», a typographic inconsistency that does not affect functionality but reduces readability. Fourth, no unit tests are present in the analyzed codebase, making refactoring risky; adoption of JUnit 5 with test data generators for numerical solution verification against the analytical `PBMALinear` baseline is recommended.

4.4. Future Development Directions

The architecture is amenable to several non-breaking extensions. Parallel computation of the inner integration loops in `U3` could be implemented through a `ParallelPBMBase` subclass using Java's `ExecutorService`. Memoization of expensive `P1`/`P2`/`P3` evaluations at fixed time-space grid points would reduce redundant computation in the consolidation analysis integration loops. A web service layer exposing `PBMBase` via a REST API could be added independently of the core modules, since the core is already stateless with respect to computation.

5. CONCLUSIONS

This paper presented the extensible software architecture of the PBM library for modeling filtration consolidation kinetics in hierarchical porous media. The following conclusions are drawn in relation to the stated objectives:

1. The system is decomposed into four modules with a strictly unidirectional dependency graph and well-defined interface contracts. This decomposition ensures that changes in one module do not propagate to others, directly supporting maintainability.
2. Three design patterns are systematically applied: the Factory pattern in `PBMBase` enables runtime model selection without exposing instantiation details to client code; the Strategy pattern via the `Draweble` interface decouples all visualization components from all mathematical models; the Template Method pattern in abstract base classes enforces the correct computational workflow sequence while delegating solver-specific steps to subclasses.
3. A new two-level solver can be integrated in three steps: implement four abstract methods, register a type constant, and update the factory without modifying any existing class outside the Direct module. Three-level extensions require an additional `P3` method override. Parameter sets and visualization components extend independently through their respective base classes.

These findings demonstrate that established object-oriented design patterns provide effective architectural governance for domain-specific scientific computing systems. The PBM architecture serves as a concrete, documented example of how modular decomposition and pattern-driven interface design can yield a system that is simultaneously operationally effective for engineering analysis and structurally open to ongoing mathematical research.

Author Contributions:

Mykhaylo Petryk: Writing – original draft, Requirements analysis, Model conception, Software Design Methodology, Software Architecture Models, Investigation, Supervision, Funding acquisition.

Dmytro Mykhalyk: Writing – review & editing, Requirements analysis, Conceptualization, Software Implementation, Supervision, Funding acquisition.

Mykola Zayarny: Writing – original draft preparation, review & editing, Formal analysis, Conceptualization, Software Architecture Models Implementation, Funding acquisition.

Oleksiy Tsebriy: Writing – review & editing, Investigation, Code design management, Qualitative analysis and verification of results.

Acknowledgments: The authors acknowledge funding from the Ministry of Education and Science of Ukraine (Grant DI 247-22 0122U001859), Embassy of France in Ukraine, CDEFI and Université de Téchnologie de Compiègne (SSHN 163389S).

Funding: This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

Conflicts of Interest: The authors declare no conflict of interest.

Data Availability Statement: The source code of the PBM library analyzed in this study is available from the corresponding author upon reasonable request.

Declaration of Generative AI and AI-assisted technologies in the writing process: During the preparation of this work, the author(s) used GitHub Copilot (Claude Sonnet 4.6) in order to assist with structuring the manuscript and generating diagram code. After using this tool/service, the author(s) reviewed and edited the content as needed and take(s) full responsibility for the content of the publication.

Supplementary Materials/Appendix A

The appendix is an optional section that can contain details and data supplemental to the main text. All appendix sections must be cited in the main text.

REFERENCES

[1] Cao B., Zhang T., Zhang W., Wang D. (2021) Enhanced technology based for sewage sludge deep dewatering: A critical review. Water Res, 189, 116650. https://doi.org/10.1016/j.watres.2020.116650.

[2] Jiang Y., Gao F., Zhang N., Li J., Xu M., Jiang Y. (2023) Dehydration performance of municipal sludge and its dewatering conditioning methods: a review. Ind. Eng. Chem. Res, 62, pp. 11337–11357. https://doi.org/10.1021/acs.iecr.3c01553.

[3] Liu Z., Luo F., He L., Wang S., Wu Y., Chen Z. (2024) Physical conditioning methods for sludge deep dewatering: A critical review. J. Environ. Manage, 360, 121207. https://doi.org/10.1016/j.jenvman.2024.121207.

[4] Hou J., Hong C., Ling W., Hu J., Feng W., Xing Y., Wang Y., Zhao C., Feng L. (2024) Research progress in improving sludge dewaterability: sludge characteristics, chemical conditioning and influencing factors, J. Environ. Manage, 351, 119863. https://doi.org/10.1016/j.jenvman.2023.119863.

[5] Shi L., Yin X., Sun H., Pan X., Yuan Z., Cai Y. (2022) A new approach for determining compressibility and permeability characteristics of dredged slurries with high water content, Can. Geotech. J., 59, pp. 965–977. https://doi.org/10.1139/cgj-2020-0676.

[6] Vorobiev E. (2022) Dewatering of non-uniformly structured wet compacts under additional compressive pressure: Predictive model, Powder Technol, 404, 117469. https://doi.org/10.1016/j.powtec.2022.117469.

[7] Vorobiev E. (2024) Analytical and numerical modelling of the effect of filter medium resistance on the mechanical dewatering of wastes and industrial filter cakes, Chem. Eng. Res. Des., 201, pp. 108–119. https://doi.org/10.1016/j.cherd.2023.11.017.

[8] Petryk M., Vorobiev E. (2013) Numerical and analytical modeling of solid--liquid expression from soft plant materials, AIChE J., 59, pp. 4762–4771. https://doi.org/10.1002/aic.14213.

[9] Logg A., Mardal K.-A., Wells G. N. (Eds.) (2012) Automated Solution of Differential Equations by the Finite Element Method: The FEniCS Book, Springer, Berlin. https://doi.org/10.1007/978-3-642-23099-8.

[10] Weller H. G., Tabor G., Jasak H., Fureby C. (1998) A tensorial approach to computational continuum mechanics using object-oriented techniques. Computers in Physics, 12, pp. 620–631. https://doi.org/10.1063/1.168744.

[11] MATLAB Documentation: Toolbox Development Guidelines, The MathWorks Inc., Natick, MA, 2023. https://www.mathworks.com/help/matlab/matlab_oop/.

[12] Kelley C. (2022) Solving Nonlinear Equations with Iterative Methods: Solvers and Examples in Julia, SIAM, Philadelphia. https://doi.org/10.1137/1.9781611977271.

[13] Petryk M., Lebovka N., Mykhalyk D., & Vorobiev E. (2026) Mechanical dewatering of wet compacts containing binary system of microporous particles. Separation and Purification Technology, vol. 382, part 3, 135775. https://doi.org/10.1016/j.seppur.2025.135775.

[14] Lebovka N., Petyk M., Vorobiev E. (2022) Monte Carlo simulation of dead-end diafiltration of bidispersed particle suspensions. Physical Review E, vol. 106, 064610. DOI: 10.1103/PhysRevE.106.064610

[15] Kelly D., Sanders C. (2008). The Challenge of Testing Scientific Software, in: Proceedings of the Testing Academic and Industrial Conference – Practice and Research Techniques, Windsor, UK, pp. 95–98.

[16] Carver J., Hong N. P. C., Thiruvathukal G. K. (Eds.) (2016). Software Engineering for Science, CRC Press, Boca Raton.

[17] Heroux M., Willenbring J. (2009). Barely sufficient software engineering: 10 practices to improve your CSE software, in: Proceedings of the 2009 ICSE Workshop on Software Engineering for Computational Science and Engineering, Vancouver, 2009, pp. 15–21.

[18] Gamma E., Helm R., Johnson R., Vlissides J. (1994). Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, Reading, MA.

[19] Press W. H., Teukolsky S. A., Vetterling W. T., Flannery B. P. (2007). Numerical Recipes: The Art of Scientific Computing, 3rd ed., Cambridge University Press, Cambridge, 2007.

[20] Martin R. C. (2017). Clean Architecture: A Craftsman's Guide to Software Structure and Design, Prentice Hall, Upper Saddle River, NJ.

[21] Fowler M., (2004). Inversion of Control Containers and the Dependency Injection Pattern, martinfowler.com. https://martinfowler.com/articles/injection.html (accessed: May 2026).

[22] Bass L., Clements P., Kazman R. (2021). Software Architecture in Practice, 4th ed., Addison-Wesley, Boston, 2021.

[23] Fowler M. (2002). Patterns of Enterprise Application Architecture, Addison-Wesley, Boston.

[24] Bourque P., Fairley R. E. (Eds.) (2014). SWEBOK: Guide to the Software Engineering Body of Knowledge, Version 3.0, IEEE Computer Society. https://www.computer.org/education/bodies-of-knowledge/software-engineering.

[25] Petryk M., Mykhalyk D. (2010) Nonlinear Mathematical Model of Two-Level Transfer of the "Filtration-Consolidation" Type. Journal of Automation and Information Sciences, 42, pp. 59–68. https://doi.org/10.1615/JAutomatInfScien.v42.i3.60.

[26] Mykhalyk D. (2011). Mathematical modeling of diffusion mass transfer in catalytic media of microporous structure particles (PhD Thesis), Ternopil National Technical University, Ternopil, http://elartu.tntu.edu.ua/handle/123456789/1381.

[27] Petryk M., Mykhalyk D. (2009) Mathematical modeling of adsorption nonlinear mass transfer in catalytic porous media, Bulletin of TNTU. 14, pp. 136–145. http://elartu.tntu.edu.ua/handle/123456789/514.

[28] Crank J. (1975). The Mathematics of Diffusion, 2nd ed., Oxford University Press, Oxford.

[29] Logg A., Mardal K.-A., Wells G. N. (Eds.) (2012). Automated Solution of Differential Equations by the Finite Element Method: The FEniCS Book, Springer, Berlin. https://doi.org/10.1007/978-3-642-23099-8.

[30] Weller H. G., Tabor G., Jasak H., Fureby C. (1998). A tensorial approach to computational continuum mechanics using object-oriented techniques. Computers in Physics, 12, pp. 620–631. https://doi.org/10.1063/1.168744.

[31] MATLAB Documentation: Toolbox Development Guidelines, The MathWorks Inc., Natick, MA, 2023. https://www.mathworks.com/help/matlab/matlab_oop/.

[32] Kelley C. Solving Nonlinear Equations with Iterative Methods: Solvers and Examples in Julia, SIAM, Philadelphia, 2022. https://doi.org/10.1137/1.9781611977271.

[33] Kelly D., Sanders C. (2008). The Challenge of Testing Scientific Software, in: Proceedings of the Testing Academic and Industrial Conference – Practice and Research Techniques, Windsor, UK, pp. 95–98.

[34] Carver J., Hong N. P. C., Thiruvathukal G. K. (Eds.) (2016). Software Engineering for Science, CRC Press, Boca Raton.

[35] Heroux M., Willenbring J. (2009). Barely sufficient software engineering: 10 practices to improve your CSE software, in: Proceedings of the 2009 ICSE Workshop on Software Engineering for Computational Science and Engineering, Vancouver, pp. 15–21.

[36] Gamma E., Helm R., Johnson R., Vlissides J. (1994). Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, Reading, MA.

[37] Press W. H., Teukolsky S. A., Vetterling W. T., Flannery B. P. (2007). Numerical Recipes: The Art of Scientific Computing, 3rd ed., Cambridge University Press, Cambridge,.

[38] Martin R. C. (2017). Clean Architecture: A Craftsman's Guide to Software Structure and Design, Prentice Hall, Upper Saddle River, NJ.

[39] Fowler M. (2004). Inversion of Control Containers and the Dependency Injection Pattern, martinfowler.com. https://martinfowler.com/articles/injection.html (accessed: May 2026).

[40] Bass L., Clements P., Kazman R. (2012). Software Architecture in Practice, 4th ed., Addison-Wesley, Boston, 2021.

[41] Fowler M. (2002). Patterns of Enterprise Application Architecture, Addison-Wesley, Boston.

[42] Bourque P., Fairley R. E. (Eds.) 2014. SWEBOK: Guide to the Software Engineering Body of Knowledge, Version 3.0. IEEE Computer Society. https://www.computer.org/education/bodies-of-knowledge/software-engineering.