Syllabus for the Course "Algorithmization and Programming Languages

Volume: 9 ECTS credit (324 hours)
Semesters: 1-4

1. General characteristics of the discipline

This course provides a comprehensive, tool-oriented foundation for the program, aimed at developing algorithmic, architectural, and systems thinking through solving software engineering problems. Mastering Python, C#, JavaScript, and DevOps practices is carried out in the logic of system task formulation and implementation using LLM as an intelligent tool.

2. Place of the discipline in the program structure

The discipline belongs to the basic part of the general science module and is implemented over four semesters. It is an instrumental basis for the disciplines of systems analysis, information systems architecture, artificial intelligence, big data, and project activities.

3. Objectives of mastering the discipline

- develop an engineering approach to algorithmization and programming;
- mastering Python, C#, and JavaScript in the context of systems engineering;
- develop LLMs application skills for analysis, generation and architectural design of software solutions;
- preparation for the implementation of complex interdisciplinary projects.

4. Objectives of the discipline

- formalization of engineering tasks and their algorithmization;
- designing the architecture of software systems;
- Integration of Git and DevOps practices;
- use LLM for analysis, refactoring and design;
- foster teamwork and collaborative development skills.

5. Planned learning outcomes

Students should be able to formalize engineering tasks, design solution architectures, develop software systems in Python, C# , and JavaScript, apply DevOps tools, and use LLM to train, analyze, generate, and refactor code.

6. Methodological concept of the discipline

Learning is structured around meaningful software engineering tasks via LLM. Each topic block begins with a system analysis of the problem, its decomposition and architecture design, after which the software implementation is performed. LLM is used as a tool for requirements formalization, architectural analysis, and engineering reflection.

6.1. Logic of the discipline's end-to-end engineering trajectory

The discipline is implemented as a consistent engineering trajectory aimed at developing professional thinking of a software system developer — from mastering individual tools to designing and implementing a comprehensive information product in a team environment.

Educational logic is based on the principle of step-by-step complication of the level of abstraction and responsibility:

Stage 1. Mastering the tool and algorithmic thinking (1 semester)

At the initial stage, training focuses on developing algorithmic thinking and skills in formalizing engineering problems. Mastering the Python language is carried out through the decomposition of tasks, analysis of alternative implementations, and formation of primary architectural representations.

LLM is used as an intelligent tool for learning, algorithm analysis, and reflection.

The result of this stage is students can to formalize the problem and implement the correct software solution with justification of the chosen algorithm.

Stage 2. Architectural design of an applied system (2nd semester)

In the second stage, the focus shifts from algorithms to software architecture. Mastery of C# and .NET is achieved through building application services, emphasizing modularity, responsibility distribution, testability, and operational resilience.

The student learns:

- designing a domain model.
- building an architectural framework.
- integration of components.
- test procedure for the software system.

LLM is used as a tool for architectural analysis, test case generation, and engineering peer review.

The result of this stage is the ability to design and implement an architecturally sound software system.

Stage 3. Subsystem integration and Information product design (3rd semester)

The third stage is aimed at developing systematic thinking. Learning JavaScript is not considered in isolation, but in the context of client architecture and subsystem integration. The key becomes:

- approval of API contracts.
- integration of client and server logic;
- analysis of the evolution of architecture;
- justification of the technology stack;
- reviewing the life cycle of an information product.

The student starts working not with a separate application, but with a complex system that interacts with the external environment.
LLM is used as a tool for system analysis, identification of architectural risks, and engineering reflection.
The result of this stage is the ability to design a comprehensive information system, taking into account its development and maintenance.

Stage 4: Team Engineering and Product lifecycle (4th semester)
The final stage is implemented in the format of a team integration project with clearly distributed roles and personal responsibility of participants.
The project covers the solution lifecycle:

- creating an idea and setting a task.
- architectural design and risk analysis;
- implementation of modules and their integration;
- conducting functional and integration tests;
- preparation of engineering documentation;
- defense of architectural decisions.

The student works in conditions of distributed responsibility (architect, developer, quality engineer, etc.), which simulates a real professional environment.
LLM is used as an intelligent tool for auditing, opposing, and preparing for engineering defense.
The result of this stage is the ability not only to implement a software system, but also to justify architectural solutions in a reasoned manner, to demonstrate their stability and readiness for commissioning.

7. Educational technologies
The AI-augmented learning model is used. Students interact with LLM in the form of a training dialogue, a master class, and an engineering assistant. LLM is used to explain syntax, analyze algorithms, generate solutions, refactor code, design architecture, and plan projects.

8. Differentiated assessment model and the principle of supportive ambition

The discipline is delivered under conditions of high expectations for students' engineering thinking and professional responsibility. This approach takes into account the variation in prior experience in students' prior training levels and rates of professional growth.
Assessment follows the principle of supportive ambition, which:

- maintains high targets for learning outcomes;
- avoids artificial standardisation of requirements;
- ensures inclusivity across different competence levels;
- fosters individual professional growth within a unified learning space.

8.1. Unified educational field

Training is carried out in a general academic environment without forming separate subgroups according to the level of training. All students work with the same tasks, architectural cases and projects.
Differentiation is achieved not by simplifying the content, but by varying the depth of study and the degree of independence of decisions.

8.2. Level structure of the educational result

Each module includes three levels of content development:
Basic level (mandatory minimum)
Student:

- implements the task correctly.
- demonstrates an understanding of the solution architecture.
- able to explain the decisions made.

Achieving the basic level is a prerequisite for successful mastery of the discipline.

Advanced level
Student:

- analyzes alternative solutions.
- justifies architectural trade-offs in a well-reasoned manner.
- identifies and eliminates implementation risks.
- demonstrates the solution's resilience to changing requirements.

Research level
Student:

- offers optimizations and architectural improvements.
- performs a comparative analysis of technologies.
- Critically evaluates solutions, including LLM's proposals.
- demonstrates independent engineering reflection.

8.3. Evaluation principles

Evaluation is based on a combination of the following components:

1. **Engineering correctness of the solution** (compliance with requirements).
2. **Depth of architectural analysis**.
3. **The degree of independence and validity of decisions**.
4. **Ability to reflect and make a reasoned defense**.

Thus, a student who reaches the basic level gets a positive final grade, and a higher grade requires a demonstration of an advanced or research level.

8.4. Differentiation in teamwork

In the 4th semester, the assessment includes:

- individual responsibility for the project module;
- analysis of your personal contribution;
- individual interview by area of responsibility;
- team defense of the integration solution.

The high level of one participant does not compensate for the insufficient contribution of the other.

8.5. Support for professional growth

The teaching strategy is focused on:

- to encourage intellectual risk-taking and initiative;
- support for switching from basic to advanced levels.
- building a culture of engineering integrity;
- targeted support for students facing academic challenges.

Calendar and thematic plan (1 semester)

Key:

Lecture (Lect.) — theoretical material delivery.
Practical / Lab. — hands-on activities, simulations and discussions.
Independent study (IS) — self-directed work on project tasks and analysis.
LLM use — applying Large Language Models for analysis, generation and decision support (integrated into independent study tasks).

Module 1. Algorithmization and tool programming (Python)

The module is aimed at forming the algorithmic and instrumental basis of software engineering.

Know: principles of algorithmization, basic data structures, fundamentals of the procedural and object approach, the role of LLM in problem formalization.
Be able to: decompose an engineering problem, develop algorithms in Python, analyze alternative implementations and their complexity.
Possess: skills in code development and debugging, basic testing, refactoring, and using LLM as an analysis and training tool.

Week	Theme	Lect. (h)	Prac. (h)	Lab. (h)	IS (h)
1-4	**Fundamentals of the Python language: syntax, basic data types, control constructs. Algorithms in Python. Technique of applying LLM in the training assistant mode.**				
1	Introduction to the course: software engineering problem statement, Git/GitHub, development environment. LLM as a learning assistant: formalizing text problems into algorithmic solutions	2	2-4	–	4
2	Python syntax. Data types. Variables. Practice of transitions between LLM modes: lecturer-mentor. Code modification based on tasks of the smart assistant with analysis of understanding goals in the dialog.	2	2	2	4
3	Conditional constructions. Branching logic. Analysis of algorithm logic in a dialog with LLM. Comparison of alternative algorithm implementations in the code.	2	2-4	–	4
4	Loops and iterations. Algorithms for processing sequences. Comparison of alternative algorithm implementations in the code in a dialog withLLM. Analysis of algorithm complexity. The concept of optimization	2	2	2	4
5-8	**Analysis of algorithms**				
5	Functions. Decomposition of the problem. Human-implemented decomposition with motivation forLLM. Setting a task for performingLLM with an understanding of the logic in the dialog.	2	2-4	–	4
6	Python data structures (list, dict, set): analysis of use cases, trade-offs, and architectural implications. Practical tasks: designing data structures for specific scenarios. LLM-facilitated debate on optimal choices.	2	2	2	4

Week	Theme	Lect. (h)	Prac. (h)	Lab. (h)	IS (h)
7	Error and exception handling. Analysis of common errors.LLM in Analysis tool mode. Generate debugging scripts.	2	2-4	–	4
8	Working with files and I / O – understanding the essence in a dialog withLLM Designing a file data handler with LLM.	2	2	2	4
9-12	**Architectural Assistant (OOP)**				
9	Introduction to OOP: Classes and Objects. A procedural solution and an object model. Direct task-LLMlecturer. The inverse problem isthe LLMexaminer. Getting a training task fromLLM. Discussion about the requirements for the solution. Implementing a solution that meets the requirements via LLM.	2	2-4	–	4
10	Class encapsulation and methods. OOP code refactoring via LLM dialog	2	2	2	4
11	Inheritance and polymorphism. alternative architectural models of classes. Direct task-LLMlecturer. Inverse problem-LLMexaminer.	2	2-4	–	4
12	Designing an object model (mini-case). Discussion about the requirements for the solution. Implementation of requirements decomposition into classes via LLM.	-2	2	2	6
13-14	**Implementation of testing, code analysis and refactoring**				
13	Testing (pytest). Basics of unittests. Direct task-LLMlecturer. The inverse problem isthe LLMexaminer. Generate test cases, analyze coverage	2	2	2	4
14	Refactor and improve code readability. Direct task-LLMlecturer. The inverse problem isthe LLMexaminer. Practical implementation:code review, design and implementation of modifications throughLLM.	-2	2	2	6
15-16	**Engineering partner**				
15	Mini-project: implementation of an engineering task. Requirements analysis, architecture and logic development. Practicum. setting a task for developing	-2	2	2	8

Week	Theme	Lect. (h)	Prac. (h)	Lab. (h)	IS (h)
	a software solution. Testing and acceptance of the implemented solution fromLLM. The idea of a role agent is to generate test data for various situations and scenarios. Target interference generation for code via LLM.				
16	Concept of software solution testing methodology. Generate a test methodology program and test scenarios via LLM. Implementation of tests of a software solution in Python through the agent architecture. Analysis of alternatives and reflection in the mode of a detailed analytical dialogue human-intelligent assistant.	–	2	-	6

Calendar and thematic plan (2nd semester)

Module 2. Architectural design of an application system (.NET / C#)

The module develops architectural thinking and instrumental skills in building application systems.

Know: principles of modular architecture, domain model design, Web API organization, and testing.

Be able to: design and implement service applications in C# / .NET, justify architectural decisions, and integrate components.

Possess: design, testing, and refactoring tools, Git and CI practices, and the use of LLM for architectural analysis and engineering peer review.

Week	Theme	Lect. (h)	Sem. (h)	Lab. (h)	IS (h)
1-5	**C# as a tool for engineering modeling**				
1	C# and .NET as an engineering development environment. The ecosystem .NET, solution and project structure, coding standards, engineering workflow organization. LLM acts as a lecturer and methodologist, contributing to the formalization of software system requirements and structuring engineering activities.	2	2	0	4
2	A typical C# system and developing a domain model. Collections, LINQ, principles of immutability, data structure design. The laboratory work is devoted to the develop a basic domain model and data processing	2	2	2	4

Week	Theme	Lect. (h)	Sem. (h)	Lab. (h)	IS (h)
	operations. LLM is used as an examiner and primary code review tool.				
3	Object-oriented design and SOLID principles. Building an architecturally stable domain model, justifying the boundaries of responsibility of components. The practical lesson is held in the format of an engineering discussion with LLM in the role of an architect-consultant.	2	2	0	4
4	Contractuality, validation, and error handling. Creating a stable application layer, designing negative scenarios, and analyzing fault tolerance. Laboratory work: implementing a service layer with exception handling. LLM acts as a generator of interference and test scenarios.	2	2	2	4
5	Asynchrony and concurrent execution in .NET. Async/await model, task management, typical design errors. The practical lesson focuses on analyzing asynchronous code defects using LLM as an engineering diagnostic tool.	2	2	0	4
6-10	**Architecture and infrastructure .NETApplications**				
6	Application architecture and dependency injection. Layered structure, separation of responsibilities, configuration mechanisms. Laboratory work: creating the architectural framework of the application. LLM is used as an architect-opponent when evaluating decisions made.	2	2	2	4
7	Designing the Web API as an engineering system. Contract formation, REST principles, justification of interaction interfaces. The practical part includes the defense of design solutions in the format of "architectural opposition" with the participation of LLM.	2	2	0	4
8	Web API implementation and infrastructure mechanisms. Middleware, centralized error handling, logging. The laboratory work is aimed at developing the service's application interface. LLM is used as a tool for analyzing code and identifying architectural risks.	2	2	2	4

Week	Theme	Lect. (h)	Sem. (h)	Lab. (h)	IS (h)
9	Data access and storage design. Using EF Core, migrations, model configuration, transactionality. The practical lesson is devoted to discussing alternatives to designing a data warehouse with LLM as an expert consultant.	2	2	0	4
10	Data access organization patterns and architecture testability. Repository, Unit of Work, analysis of the applicability of patterns. Lab work: implementing a data access layer and integration tests. LLM acts as an opponent in the defense of architectural choice.	2	2	2	4
11-16	System quality, safety and operational maturity				
11	Engineering testing strategy. Approaches to unit and integration testing, formation of test scenarios, analysis of coverage completeness. Practical training is conducted using LLM as a test set generator and examiner.	2	2	0	4
12	Observability and operational stability. Logging, metrics, tracing, and health monitoring. The laboratory work is aimed at implementing monitoring and diagnostic tools. LLM is used for analyzing operational risks and logging correctness.	2	2	2	4
13	Application system security and access control. Basic authentication and authorization mechanisms, threat analysis, configuration and data protection. The practical part involves conducting simplified threat-modeling with the participation of LLM as an expert auditor.	2	2	0	4
14	Refactoring and maintainability of the software system. Analyze architectural defects, improve code structure, and evaluate technical debt. Laboratory work includes conducting an AI-assisted code review and a reasoned defense of the changes made.	2	2	2	4
15	Preparing the software system for release. Organization of assembly, automation of testing, formation of readiness criteria. The practical lesson is aimed at designing a CI process with a critical assessment of LLM recommendations.	2	2	0	4

Week	Theme	Lect. (h)	Sem. (h)	Lab. (h)	IS (h)
16	Component integration and software system testing methodology. Comprehensive verification of the implemented service, develop the program and test methods, analysis of the stability of architectural solutions. Laboratory work includes demonstration of results and engineering reflection with LLM as an opponent.	2	2	2	4

Calendar and thematic plan (3rd semester)

Module 3. Client architecture and subsystem integration in Information Systems (JavaScript)

The module is aimed at developing system and integration thinking in the develop information systems. JavaScript is considered as a tool for building a client architecture as part of a comprehensive software solution.

Know: principles of client and component architecture, API contracts, subsystem integration, and evolution of architectural solutions.

Be able to: design the client part of the system, ensure consistency of interaction with the server subsystem, justify the choice of the technology stack, taking into account the product life cycle.

Possess: tools for modular and team development, integration testing, architectural analysis, and LLM usage for system risk assessment and engineering reflection.

Week	Theme	Lect. (h)	Sem. (h)	Lab. (h)	IS (h)
1-6	**JavaScript as a client architecture tool**				
1	JavaSJavaScript as a client architecture tool. Execution model, asynchrony, and runtime features. System analysis of client and server logic differences using LLM as an architectural mapping tool.	2	2	0	4
2	Modular organization of the client application. Interface decomposition, project structuring, and distribution of component responsibilities. Laboratory work: designing the client application architecture using LLM to analyze alternative solutions.	2	2	2	4
3	Asynchronous communication and network request processing. Designing a stable client layer.	2	2	0	4

Week	Theme	Lect. (h)	Sem. (h)	Lab. (h)	IS (h)
	Engineering analysis of client–server interaction scenarios in a dialog with LLM.				
4	Application state management. Architectural approaches to data storage and synchronization. Laboratory work: implementing a state management model and evaluating its stability using LLM.	2	2	2	4
5	Component-based user interface architecture. UI decomposition, matching user logic and business rules. Practical justification of the chosen model in the format of an engineering discussion.	2	2	0	4
6	Typing and engineering stability of client code. Conceptual foundations of TypeScript. Laboratory work: client code refactoring with justification for improving reliability and maintainability.	2	2	2	4
7-11	**Integration of subsystems and coordination of architectural solutions**				
7	API contract as a basis for system integration. Formation of interaction agreements, coordination of data models. Practical lesson in the format of architectural opposition using LLM.	2	2	0	4
8	Integration of client and server subsystems. Modeling of negative scenarios and fault tolerance analysis. Laboratory work: implementation and testing of integration interaction.	2	2	2	4
9	Versioning and architecture evolution. Maintainability, compatibility of changes, technical debt. Analytical reflection using LLM as a risk assessment tool.	2	2	0	4
10	Engineering organization of team development. Code review procedures, fixing architectural decisions, and approving changes. Laboratory work: modeling an architectural review.	2	2	2	4
11	Integration mini-project: problem statement and decomposition. Formalizing requirements and assigning responsibility to components. LLM is used to clarify requirements and identify contradictions.	2	2	0	4

Week	Theme	Lect. (h)	Sem. (h)	Lab. (h)	IS (h)
12-16	**Integrated information product architecture**				
12	Implementation and coordination of an integration solution. Laboratory work: integration of modules and analysis of correct interaction.	2	2	2	4
13	Designing a comprehensive information product. System analysis of the task, determination of constraints and success criteria.	2	2	0	4
14	Architectural justification of the technology stack. Comparison of alternative solutions and arguments for choosing development tools. Laboratory work: preparation of the architectural scheme of the system.	2	2	2	4
15	Information product lifecycle and engineering tradeoffs. Link between architectural solutions and the organizational development model (with an interdisciplinary reference to the project management course).	2	2	0	4
16	Integrated system testing methodology and engineering reflection. Creating a test program, analyzing the sustainability of architectural solutions, defending project work in the format of an analytical dialogue using LLM.	2	2	2	4

Calendar and thematic plan (4th semester)
Module 4. Team Engineering and Software solution lifecycle
(4th semester)
The module is implemented in the format of a team integration project.
Know: stages of the software product life cycle, principles of distributed responsibility and architectural project management.
Be able to: formalize requirements, design an architecture, implement and integrate modules, conduct tests, and defend solutions in a reasoned manner.
Possess: tools for team development, engineering documentation, integration testing, and using LLM as a means of auditing, analyzing, and preparing for professional defense.

Week	Theme	Lect. (h).	Prac. (h)	Lab. (h)	IS (h)
1-4	**Engineering statement and architectural justification of the project**				
1	Formation of the project team and engineering statement of the problem. Definition of the subject area, formalization of requirements and distribution of areas of responsibility of project participants.	0	2	2	6
2	Architectural design of system modules. Develop a block diagram, defining interfaces and contracts for interaction of components.	0	2	2	6
3	Coordination of architectural solutions and risk analysis. Engineering consultation, identification of contradictions and clarification of the integration model.	0	2	2	6
4	Individual defense of the architectural part of the project. Justification of the decisions made and analysis of alternatives.	0	2	2	6
5-8	**Implementation and integration verification of the architecture**				
5	Implementation of the architectural framework and distributed modules. Creating the basic structure of the software system.	0	2	2	6
6	Functionality development and local component testing. Checking whether implemented modules meet the requirements.	0	2	2	6
7	Component integration and architecture adjustment. Identification of incompatibilities and improvement of interfaces.	0	2	2	6
8	Functional and integration tests of the system. Failure analysis, stability testing, and acceptance criteria compliance.	0	2	2	6
9-10	**Engineering maturity and solution defense**				
9	Preparing engineering documentation and fixing architectural solutions. Description of the implemented system and the contribution of each participant.	0	2	2	6
10	Team project defense and individual interview. Demonstration of the system, reasoning of architectural solutions, engineering reflection.	0	2	2	