Національний технічний університет України
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Емблема кафедри (за наявності)

Department of Computer Engineering

Software Engineering

Syllabus

Requisites of the Course

Cycle of Higher Education	***First cycle of higher education (Bachelor's degree)***
Field of Study	*12 Information Technologies*
Speciality	*123 Computer Engineering*
Education Program	*Computer Systems and Networks*
Type of Course	*Normative*
Mode of Studies	*full-time*
Year of studies, semester	*2nd year (3rd semester)*
ECTS workload	*5 credits (ECTS). 150 hours,*
Testing and assessment	*3 semester – Exam,*
Course Schedule	*2 classes per week by the timetable http://rozklad.kpi.ua/*
Language of Instruction	*English*
Course Instructors	Lecturer: PhD, Associate Professor, Andrii Antoniuk, mobile +380503588019, email antoniuk.andrii@lll.kpi.ua, personal web page http://se-111.blogspot.com Teacher of practical work: PhD, Associate Professor, Andrii Antoniuk, mobile +380503588019, email antoniuk.andrii@lll.kpi.ua, personal web page http://se-111.blogspot.com Teacher of laboratory work: PhD, Associate Professor, Andrii Antoniuk, mobile +380503588019, email antoniuk.andrii@lll.kpi.ua, personal web page http://se-111.blogspot.com
Access to the course	BigBlueButton, Cisco, Google, TeamViewer, Zoom, http://se-111.blogspot.com

Outline of the Course

1. Course description, goals, objectives, and learning outcomes

The purpose *of studying the credit module "Software Engineering" is to provide students with practical skills in designing and building programs for computer systems:*
- study the principles of design and construction of software for computer systems
- study of the template approach to the design of software systems and UML language
- study of methods and tools of software development
- study of object-oriented approach to software development.
The subject *of the credit module "Software Engineering" are - methods, techniques, templates, tools and systems for research, automated and automatic design, commissioning, production and operation, design documentation, standards, procedures and tools to support software lifecycle management.*
Students after mastering the credit module must demonstrate the following program ***learning outcomes****:*
knowledge:
- stages, methods and standards of software development;
- template design method;
- basic software design templates and principles of their construction and application,
- principles of component design.
skill:
- identify the sources of requirements and ensure the process of their formation;

- develop specifications of user requirements;
- analyze the requirements;
- develop a specification of software requirements;
- model different aspects of the system;
- design the components of the architectural solution;
- apply and create reusable components.
experience:
- template design method;
- development of specifications of user requirements;
- modeling of various aspects of the system;
- application and creation of reusable components.

2. Prerequisites and post-requisites of the course (the place of the course in the scheme of studies in accordance with curriculum)

The list of disciplines, or knowledge and skills that the student needs:
- basic level of knowledge of Java programming language, or JavaScript, or C ++, or Python;
- basic level of mastery of the tools of the integrated development environment Eclipse or IntelliJ IDEA;
- basic level of mastery of UML simulation tools ArgoUML or Umbrello.
The following disciplines are based on the learning outcomes of this discipline:
- course work on Software Engineering;
- organization of computational processes;
- basics of parallel programming;
- parallel and distributed calculations.

3. Content of the course

List of sections and topics of the whole discipline:
- Section 1. Software development methodologies:
  - Topic 1.1. Basic concepts and problems of software development.
  - Topic 1.2. Software life cycle; international software life cycle standards.
  - Topic 1.3. Models and methodologies of software development.
  - Topic 1.4. Analysis, specification, verification and validation of software requirements. Functional and non-functional requirements.
- Section 2. Software development tools and environments:
  - Topic 2.1. Integrated software development environments.
  - Topic 2.2. Project management systems (Redmine, JIRA).
  - Topic 2.3. Document version control systems, architectural features (CVS, SVN, Git).
  - Topic 2.4. Project build automation tools (make utility, CMake, Ant and Maven systems).
- Section 3. Software design using templates:
  - Topic 3.1. Software architecture design.
  - Topic 3.2. Software design templates. Classification of design templates. Graphic notation.
  - Topic 3.3. Structural design templates Composite, Decorator, Proxy.
  - Topic 3.4. Structural design templates Flyweight, Adapter, Bridge, Facade.
  - Topic 3.5. Behavior patterns Iterator, Mediator, Observer.
  - Topic 3.6. Behavior patterns Strategy, Chain of Responsibility, Visitor.
  - Topic 3.7. Behavior patterns Memento, State, Command, Interpreter.
  - Topic 3.8 Generating design templates Prototype, Singleton, Factory Method.
  - Topic 3.9. Generating design templates Abstract Factory, Builder.
- Section 4. User interface design:
  - Topic 4.1. Purpose and types of user interfaces.
  - Topic 4.2. Principles of building a graphical user interface.
  - Topic 4.3. Event models. Graphical user interface drawing models.
- Section 5. Software modeling:
  - Topic 5.1. Modeling methodologies SADT, IDEF, DFD, ELM, OOAD. Modeling languages.

- *Topic 5.2. Information modeling. Entity-relationship diagrams, classes.*
- *Topic 5.3. Modeling of business processes, organizations and goals.*
- *Topic 5.4. Behavioral modeling. Diagrams of states, activities, interactions, sequences, time.*
- *Topic 5.5. Structural modeling.*
- *Topic 5.6. Functional modeling.*
- *Topic 5.7. Modeling of data flows.*
- *Topic 5.8. Modeling automation tools (ERWin, BPWin, Enterprise Architect and others).*

- Section 6. Methods of software quality assurance and control:

- *Topic 6.1. Software quality; software quality metrics and standards.*
- *Topic 6.2. Software verification and validation.*
- *Topic 6.3. Software testing.*
- *Topic 6.4. Principles of constant software integration.*
- *Topic 6.5. Code optimization and refactoring.*
- *Topic 6.6. Software performance aspects.*
- *Topic 6.7. Testing process automation tools (JUnit, JMeter).*
- *Topic 6.8. Continuous integration servers (Hudson, CruiseControl).*

- Section 7. Software project management:

- *Topic 7.1. Project management tasks. Triangle of constraints.*
- *Topic 7.2. Project content and quality management.*
- *Topic 7.3. Resource management. Project schedule planning.*
- *Topic 7.4. Risk management of the software project.*
- *Topic 7.5. Configuration and change management.*
- *Topic 7.6. Control and monitoring of the project status. Control metrics.*
- *Topic 7.7. Organization of the project team. Roles and areas of responsibility of team members.*

- Coursework.

4. Coursebooks and teaching resources

Basic literature:

1. *Design Patterns: elements of reusable object-oriented software / Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. Indianapolis: - Addison-Wesley, 1994. - 417 p. ISBN: 0201633612.*
2. *Mark Grand. Patterns in Java: A Catalog of Reusable Design Patterns Illustrated with UML, Volume 1, 2nd Edition. - Wiley Publishing, 2002. - 480 p. ISBN: 0471258393*
3. *Theo Mandel. The Elements of User Interface Design. - John Wiley & Sons, 2005. - 468 p. ISBN: 0471162671*
4. *Bruce Eckel. Thinking in Java, 4th ed. – Printice hall, 2006. – 1057 p. ISBN: 0131872486*

Educational content

5. Methodology

Lectures

№ lectures	*The title of the lecture topic and a list of main issues: list of teaching aids, references to literature and tasks for independent work of students (IWS)*
1.	Software life cycle. Basic concepts and problems of software development. Software life cycle; international software life cycle standards [1, p. 12-14; 2, p. 41-60] IWS: ISO standards are related to software development
2.	Models and methodologies of development. Models and methodologies of development. Analysis, specification, verification and validation of software requirements. Functional and non-functional requirements [1, p. 12-14; 2, p. 41-60] IWS: Flexible development methodologies

№ lectures	*The title of the lecture topic and a list of main issues: list of teaching aids, references to literature and tasks for independent work of students (IWS)*
3.	Integrated software development environments. Integrated software development environments. Project management systems (Redmine, JIRA). [1, p. 15-45; 2, p. 19-40; 3, p. 5-7] IWS: Installation and use of project management systems
4.	Version control and software assembly automation. Document version control systems, architectural features (CVS, SVN, Git). Project build automation tools (make utility, CMake, Ant and Maven systems). [1, p. 15-45; 2, p. 19-40; 3, p. 5-7] IWS: Installing and configuring SVN. Using Ant to work with SVN
5.	Software design patterns. Software architecture design. Software design patterns. Classification of design patterns. Graphic notation [1, p. 140-141, 162-173] IWS: GRASP design patterns
6.	Structural design patterns: Composite and Decorator. Composite pattern: motivation, structure, participants, relationships, ways of application and result of use. Example of implementation of the Composite pattern. Decorator pattern: motivation, structure, participants, relationships, ways of application and result of use. Example of implementation of the Decorator pattern. [1, p. 173-183, 203-213; 3, p.19-22] IWS: Composite and Decorator implementations in Java SDK classes
7.	Structural design patterns: Proxy and Flyweight. Proxy pattern: motivation, structure, participants, relationships, ways of application and result of use. Example of Proxy pattern implementation. Flyweight pattern: motivation, structure, participants, relationships, ways of application and result of use. Example of implementation of the Flyweight pattern. [1, p. 191-203, 141-152; 3, pp.19-22, 29-33] IWS: Proxy and Flyweight implementations in Java SDK classes
8.	Structural design patterns: Adapter, Bridge and Facade. Adapter pattern: motivation, structure, participants, relationships, ways of application and result of use. Example implementation of the Adapter pattern. Bridge pattern: motivation, structure, participants, relationships, ways of application and result of use. Example of implementation of the Bridge pattern. Facade pattern: motivation, structure, participants, relationships, ways of application and result of use. Example of implementation of the Facade pattern. [1, p. 152-162, 183-191; 3, p.29-33] IWS: Adapter, Bridge, and Facade implementations in Java SDK classes
9.	Behavioral patterns: Iterator and Mediator. Assignment of Behavioral patterns. Iterator pattern: motivation, structure, participants, relationships, ways of application and result of use. Example of Iterator pattern implementation. Mediator pattern: motivation, structure, participants, relationships, ways of application and result of use. Example of implementation of the Mediator pattern. [1, p. 216-217, 249-263; 4, p. 6-8] IWS: Iterator and Mediator implementations in Java SDK classes
10.	Behavioral patterns: Observer and Strategy. Observer pattern: motivation, structure, participants, relationships, ways of application and result of use. Example of implementation of the Observer pattern. Strategy pattern: motivation, structure, participants, relationships, ways of application and result of use. Example of implementation of the Strategy pattern. [1, p. 263-272, 280-291; 4, p. 6-8, 14-16] IWS: Observer and Strategy implementations in Java SDK classes

№ lectures	*The title of the lecture topic and a list of main issues: list of teaching aids, references to literature and tasks for independent work of students (IWS)*
11.	Behavioral patterns: Command and Visitor. Command pattern: motivation, structure, participants, relationships, ways of application and result of use. Example implementation of the Command pattern. Visitor pattern: motivation, structure, participants, relationships, ways of application and result of use. Example of Visitor pattern implementation. [1, p. 227-236, 300-309; 4, p. 14-16, 22-25] CPC: Command and Visitor implementations in Java SDK classes
12.	Behavioral patterns: State and Memento. State pattern: motivation, structure, participants, relationships, ways of application and result of use. Example of implementation of the State pattern. Memento pattern: motivation, structure, participants, relationships, ways of application and result of use. Example of implementation of the Memento pattern. [1, p. 272-280, 291-300, 314-328; 4, p. 22-25] IWS: State and Memento implementations in Java SDK classes
13.	Behavioral patterns: Interpreter and Chain of Responsibility. Interpreter pattern: motivation, structure, participants, relationships, ways of application and result of use. Example of implementation of the Interpreter pattern. Chain of Responsibility pattern: motivation, structure, participants, relationships, ways of application and result of use. Example of Chain of Responsibility pattern implementation. [1, p. 238-249, 217-226; 4, p. 14-16, 22-25] IWS: Interpreter and Chain of Responsibility implementations in Java SDK classes
14.	Creational patterns Prototype, Factory Method and Abstract Factory. Assignment of Creational patterns. Prototype pattern: motivation, structure, participants, relationships, ways of application and result of use. An example of the implementation of the Prototype pattern. [1, p. 89-93, 121-130; 4, p. 30-32]. Factory Method pattern: motivation, structure, participants, relationships, ways of application and result of use. Example of implementation of the Factory Method pattern. Abstract Factory pattern: motivation, structure, participants, relationships, ways of application and result of use. Example of implementation of the Abstract Factory pattern. [1, p. 111-121, 93-102; 4, p. 30-32, 37-38] IWS: Implementations of Prototype, Factory Method and Abstract Factory in Java SDK classes
15.	Singleton and Builder patterns. Singleton pattern: motivation, structure, participants, relationships, ways of application and result of use. Example of implementation of the Singleton pattern. Pattern Builder: motivation, structure, participants, relationships, ways of application and result of use. Example of implementation of the Builder pattern. [1, p. 130-138, 102-111; 4, p. 30-32, 37-38] IWS: Singleton and Builder implementations in Java SDK classes
16.	Graphical user interface. Purpose and types of user interfaces. Principles of building a graphical user interface. [5, p. 41-56] IWS: Frame and window user interfaces
17.	Event models and drawings. Event models. Graphical user interface drawing models. [5, p. 65-91] IWS: Implementation of light components on the example of SWING
18.	Information modeling and business process modeling Modeling methodologies SADT, IDEF, DFD, ELM, OOAD. Modeling languages. Information modeling. Entity-relationship diagrams, classes. Modeling of business

№ lectures	*The title of the lecture topic and a list of main issues: list of teaching aids, references to literature and tasks for independent work of students (IWS)*
	processes, organizations and goals. [6, p. 506-537] CPC: Rational Unified Process
19.	Behavioral, structural and functional modeling Behavioral modeling. Diagrams of states, activities, interactions, sequences, time. Structural modeling. Functional modeling. [6, p. 506-537] IWS: Simulation using UML
20.	Modeling of data flows Modeling of data flows. Modeling automation tools (ERWin, BPWin, Enterprise Architect and others). [6, p. 506-537] IWS: IBM modeling automation tools
21.	Software testing Software quality; software quality metrics and standards. Software verification and validation. Software testing. Testing process automation tools (JUnit, JMeter). [6, p. 368-402] IWS: Testing using JUnit. Version differences
22.	Ongoing software integration Principles of constant software integration. Code optimization and refactoring. Software performance aspects. Continuous integration servers (Hudson, CruiseControl). [6, p. 368-402] IWS: Installation and use of a continuous integration server
23.	Project management Project management tasks. Triangle of constraints. Project content and quality management. [6, p. 562-597] IWS: Means of automation of project management
24.	Resource, risk and configuration management Resource management. Project schedule planning. Risk management of the software project. Configuration and change management [6, p. 562-597] IWS: Resource, risk and configuration management tools
25.	Control and monitoring of the project status Control and monitoring of the project status. Control metrics. Organization of the project team. Roles and areas of responsibility of team members. [6, p. 562-597] IWS: Team work with a flexible methodology

Laboratory works:

The purpose of the cycle of laboratory work is for students to acquire the necessary practical skills of software development using an object-oriented approach, the use of design templates and modeling using the UML language.

Laboratory work includes:

1. *development and analysis of a mathematical algorithm for solving the problem;*
2. *decomposition of the problem;*
3. *application of design templates;*
4. *printout of the program;*
5. *UML class interaction diagrams;*
6. *the results of the program;*
7. *documentation of laboratory work and the made program*
8. *conclusions on mastering the topic of laboratory work.*

№	*Name of laboratory work*	*Number of*

laboratory work		*hours*
1.	Draft software application. Build a project with Ant. (Section 2, Topics 2.1-2.2)	2
2.	UML graphic notation. Documentation using JavaDoc. (Section 2, Topics 2.3-2.4)	2
3.	Structural software design templates. Composite, Decorator, Proxy. (Chapter 3, Topic 3.3)	2
4.	Structural software design templates - 2. Flyweight, Adapter, Bridge, Facade. (Chapter 3, Topic 3.4)	2
5.	Behavioral patterns. Iterator, Mediator, Observer. (Chapter 3, Topic 3.5)	2
6.	Behavioral patterns - 2. Strategy, Chain of Responsibility, Visitor. (Section 3, Topic 3.6)	2
7.	Behavioral patterns - 3. Memento, State, Command, Interpreter. (Section 3, Topic 3.7)	2
8.	Creational patterns. Prototype, Singleton, Factory Method. (Section 3, Topic 3.8)	2
9.	Generating templates - 2. Abstract Factory, Builder. (Chapter 3, Topic 3.9)	2
	Total:	18

6. Self-study

№ in order	*The name of the topic that is submitted for self-study (preparation for classroom classes)*	*Number of hours*
1.	ISO standards are related to software development ISO-12207, ISO-15504, ISO 9000 - ISO 9004 [Additional 3, p. 6-16; Additional 7, p.43-55]	4
2.	Flexible development methodologies [Additional 7, p.17-36, 67-83; https://uk.wikipedia.org/wiki/Gnuchka_rozrobka_programnogo_zabezpechennya]	4
3.	Installation and use of project management systems [Additional 2, p.384-441; https://uk.wikipedia.org/wiki/Управление_проектами]	4
4.	Installing and configuring SVN. Application of Ant to work with SVN [Additional 3, p.17-36; https://uk.wikipedia.org/wiki/Subversion; https://uk.wikipedia.org/wiki/Apache_Ant]	4
5.	GRASP design templates [Additional 2 p. 105-134; Additional 9 p. 299-386; https://uk.wikipedia.org/wiki/GRASP]	2
6.	Composite and Decorator implementations in Java SDK classes [1, p. 162-182; https://uk.wikipedia.org/wiki/SDK, http://www.oracle.com/technetwork/java/javase/downloads/index.html]	2
7.	Proxy and Flyweight implementations in Java SDK classes [1, p. 191-212]	2
8.	Implementations of Adapter, Bridge and Facade in Java SDK classes [1, p. 141-161, 183-190]	2
9.	Implementations of Iterator and Mediator in Java SDK classes [1, p. 249-271]	2
10.	Implementations of Observer and Strategy in Java SDK classes [1, p. 280-290, 300-308]	2
11.	Implementations of Command and Visitor in Java SDK classes [1, p. 227-235, 314-327]	2
12.	Implementations of State and Memento in Java SDK classes [1, p. 272-279, 291-299]	2
13.	Implementations of Interpreter and Chain of Responsibility in Java SDK classes [1, p. 217-226, 236-248]	2
14.	Implementations of Prototype, Factory Method and Abstract Factory in Java SDK	2

	classes [1, p. 93-101, 111-129]	
15.	Singleton and Builder implementations in Java SDK classes [1, p. 102-110, 130-137]	2
16.	Frame and window user interfaces [5, p. 61-69; https://en.wikibooks.org/wiki/Learn_Java/Graphic_User_Interface]	4
17.	Implementation of light components on the example of SWING [5 p. 8-18, 61-69; https://en.wikibooks.org/wiki/Learn_Java/Location_Managers]	4
18.	Rational Unified Process [Additional 7, p. 280-289; https://uk.wikipedia.org/wiki/Rational_Unified_Process]	2
19.	Simulation using UML [Additional 2, p. 599-602; Additional 5, p. 34-197; Additional 7, 204-246]	4
20.	IBM modeling automation tools [Additional 7, p. 443-456; https://uk.wikipedia.org/wiki/Автоматизація_процесу_програмування, http://posibniki.com.ua/post-avtomatizaciya-analizu--ta-proektuvannya--za-dopomogoyu-ibm-rational-rose, http: // easy-code.com.ua/2011/05/kerovana-modellyu-rozrobka-nastupnogo-pokolinnya-ibm-rational]	4
21.	Testing using JUnit. Differences of versions [Additional 1, p.751-765; Additional 4, pp.91-124; https://uk.wikipedia.org/wiki/Junit, https://habrahabr.ru/post/120101/]	4
22.	Installation and use of the server of constant integration [6, p. 368-402; https://en.wikipedia.org/wiki/Continuous_integration]	4
23.	Means of automation of project management [6, p. 562-597; https://uk.wikipedia.org/wiki/Управление_проектами, https://uk.wikipedia.org/wiki/Розробка_програмного_забеспечення]	4
24.	Resource, risk and configuration management tools [6, p. 562-597, https://uk.wikipedia.org/wiki/Управление_розробкой_програмного_забеспелення, https://uk.wikipedia.org/wiki/DSDM]	4
25.	Team work with a flexible methodology [6, p. 562-597, https://uk.wikipedia.org/wiki/Гнучка_розробка_програмного_забезпечення]	4
26.	In total	78

Policy and Assessment

7. Course policy

- *The student must be present at each lesson.*
- *A student may be absent from class if he / she has passed the relevant laboratory work or for a good reason: illness, etc.*
- *During the lesson, students can talk to each other only with the permission of the teacher.*
- *If necessary, use the phone, the student can quietly leave the classroom without the permission of the teacher and quietly return after the conversation on the phone.*
- *If necessary, the student can use the means of communication to search for information on the Internet, etc.*
- *Only one laboratory work is defended in one lesson. If a student wishes to submit another job, the student must stand at the end of the queue. Only 2 works can be submitted for one lesson.*
- *If necessary, you can defend the work online, for this the teacher will be allocated a separate time.*
- *Each laboratory work has its own deadline. Violation of the deadline is punishable by a fine.*
- *Reassignment of laboratory work is possible in the next lesson.*
- *Surrendering someone else's work is punishable by a fine.*

8. Monitoring and grading policy

At the first class the students are acquainted with the grading policy which is based on Regulations on the system of assessment of learning outcomes https://document.kpi.ua/files/2020_1-273.pdf

The student's rating in the course consists of points that he/she receives:

3rd semester

For laboratory works (R1), modular control works (R2) and the exam (R3).

Rs=R1+R2+R3=100 points

Each laboratory work consists of 4 parts:

protocol (2 points),

file with the text of the source code (2 points),

documentation of the program by means of IDE (2 points),

protection of laboratory work (2 points).

Total only 8 points.

In the 3rd semester 2 modular tests of 4 points each.

A student can get up to 20 points for the exam.

As a result, the maximum average weight score is equal to:

9 laboratory works x 8 points = 72 points

2 modular control works x 4 points = 8 points

Exam = 20 points

According to the university regulations on the monitoring of students' academic progress (https://kpi.ua/document_control) there are two assessment weeks, usually during 7th/8th and 14th/15th week of the semester, when students take the Progress and Module tests respectively, to check their progress against the criteria of the course assessment policy.

The students who finally score the required number of points (≥60) can:

- get their final grade according to the rating score;
- perform a Fail/Pass test in order to increase the grade.

Students whose final performance score is below 60 points but more than 30 are required to complete a Fail/Pass test. If the grade for the test is lower than the grade, which the student gets for his semester activity, a strict requirement is applied - the student's previous rating is canceled and he receives a grade based on the results of the Fail/Pass test. Students whose score is below 30 are not allowed to take the Fail/ Pass Test.

The final performance score or the results of the Fail/ Pass Test are adopted by university grading system as follows:

Score	*Grade*
100-95	Excellent
94-85	Very good
84-75	Good
74-65	Satisfactory
64-60	Sufficient
Below 60	Fail
Course requirements are not met	Not Graded

9. Additional information about the course

- *Enrollment of certificates for distance or online courses on the subject is not provided.*
- *List of questions to be submitted for semester control:*
  1. Software engineering. Definition. History.

2. What conditions must be met for the effective use of Flyweight.
3. Implement the Builder pattern.
4. XML. Appointment. Structure. Example.
5. Adapter pattern.
6. Abstract Factory pattern based on Prototype.
7. Rules to which the well-formed XML-document should correspond.
8. Duplex Adapter.
9. Implement the Abstract Factory pattern based on the Factory Method.
10. XML namespaces. Appointment. Example of use.
11. Bridge pattern.
12. Transparent Composite Patterns.
13. ANT. Appointment. Script file structure. Example script.
14. Flyweight pattern.
15. Command pattern.
16. The difference between ANT and analogues. Algorithm for creating new tasks for ANT.
17. Flyweight pattern.
18. Patterns secure Composite and internal Iterator to bypass the "depth" of hierarchical structures based on it.
19. Software life cycle. Software life cycle standards.
20. Flyweight pattern. Appointment, motivation, structure, participants. Flyweight-objects that are divided into indivisible.
21. Patterns secure Composite and Iterator-cursor to bypass the "width" of hierarchical structures based on it.
22. Model and methodology of software life cycle. Brief description of the main models of LF.
23. Facade pattern.
24. Secure Composite and Visitor patterns for representing and calculating arithmetic expressions.
25. The main processes of software life cycle.
26. Proxy pattern.
27. Strategy pattern for sorting algorithms.
28. Cascade model of software life cycle.
29. Composite pattern.
30. Chain of Responsibility pattern.
31. Iterative / incremental model of software life cycle.
32. Decorator pattern.
33. Observer pattern with update manager.
34. Spiral model of Bohemia.
35. Observer pattern.
36. State pattern with state change in context.
37. Flexible software development methodologies. General features.
38. Mediator pattern.
39. Pattern variable adapter.
40. Software requirements. Definition.
41. Iterator pattern.
42. Flyweight pattern with divisible and indivisible objects.
43. Functional and non-functional requirements. Definition. Example.
44. Visitor Pattern. Appointment, structure, participants. In which cases the use of Visitor is inappropriate. Single and double scheduling.
45. Bridge pattern.
46. User requirements.
47. Strategy Pattern. Appointment, structure, participants. Results of application and alternatives.

48. Pattern Builder. Ensure that only one instance of each particular builder exists.
49. System requirements.
50. Command pattern. Appointment, structure, participants. Compare Command and Strategy. Common and distinctive features.
51. Abstract Factory pattern based on Prototype.
52. Documentation of requirements.
53. Chain of Responsibility pattern. Appointment, structure, participants. The results of use.
54. Abstract Factory pattern based on Factory Method. How to ensure the existence of only one copy of each specific factory.
55. Development of requirements. Model, participants, management and control.
56. Observer pattern. Appointment, structure, participants. Unexpected updates. Causes and methods of neutralization.
57. Transparent Composite and Interpreter patterns for representing and calculating arithmetic expressions.
58. Formation and analysis of requirements.
59. Strategy Pattern. Appointment, structure, participants. The presence of any mechanism in the programming language removes the need for Strategy.
60. Command pattern. How to provide the ability to log and "roll back" commands.
61. Certification of requirements.
62. Visitor Pattern. Appointment, structure, participants. Compare Visitor and Internal Iterator.
63. Patterns secure Composite and internal Iterator to bypass the "depth" of hierarchical structures based on it.
64. Requirements management.
65. Iterator pattern. Appointment, structure, participants. How the client without knowing a specific unit creates an iterator that is able to work with the unit.
66. Patterns secure Composite and Iterator-cursor to bypass the "width" of hierarchical structures based on it.
67. UML. Appointment. History of creation.
68. Memento pattern. Appointment, structure, participants. Compare with alternative solutions.
69. Secure Composite and Visitor patterns for representing and calculating arithmetic expressions.
70. Types of UML-diagrams with a brief description of each. Examples.
71. State Pattern. Appointment, structure, participants.
72. Strategy pattern for sorting algorithms. How to ensure the independence of the implementation of the classes of units and elements.
73. Class diagram. Appointment, Notation. Example.
74. Pattern Method pattern. Appointment, structure, participants. The results of use. What is the pattern method?
75. Chain of Responsibility pattern. How to remove a handler and change the handler's priority by moving it in a chain.
76. The relationship of association, aggregation and composition in the class diagram. Example.
77. Interpreter pattern. Appointment, structure, participants. Results of use and alternatives. Compare with Composite.
78. Observer pattern with update manager.
79. The relationship of implementation, generalization and dependence on the class diagram. Example.
80. Singleton pattern. Appointment, structure, participants. The results of use. Compare Singleton and class with static members.
81. State pattern with state change in context. How to ensure the existence of only one copy of each specific state.
82. Roles. Multipliers. Stereotypes. Example.

83. Prototype and Factory Method patterns. Appointment, structure, participants and results. Compare their application to create an object without information about its class.
84. Variable Adapter pattern.
85. Diagram of activities. Appointment. Notation. Example.
86. Abstract Factory Pattern. Appointment, structure, participants and results. Compare different implementations of AF ..
87. Flyweight pattern with divisible and indivisible objects. Ensure the existence of only one copy of the factory.
88. Sequence diagram. Appointment. Notation. Example.
89. Pattern Builder. Appointment, structure, participants and results. Roles of pattern participants.
90. Bridge Pattern.
91. Diagram of cooperation. Appointment. Notation. Example.
92. GRASP patterns. Brief description of each pattern.
93. Pattern Builder. How to ensure the existence of only one instance of each particular builder.
94. Diagram of precedents. Appointment. Notation. Example.
95. MVC pattern. Appointment, structure, participants and results. Modifications.
96. Implement the Abstract Factory pattern based on Prototype.

Syllabus of the course

Is designed by teacher PhD, Associate Professor, Andrii Antoniuk

Adopted by Department of Computing Technics (protocol № 10, 25 May 2022)

Approved by the Faculty Board of Methodology (protocol № 10, 09 June 2022)