Software Engineering-1

Syllabus

Requisites of the Course

<table>
<thead>
<tr>
<th>Cycle of Higher Education</th>
<th>First cycle of higher education (Bachelor’s degree)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field of Study</td>
<td>12 Information Technologies</td>
</tr>
<tr>
<td>Speciality</td>
<td>123 Computer Engineering</td>
</tr>
<tr>
<td>Education Program</td>
<td>Computer Systems and Networks</td>
</tr>
<tr>
<td>Type of Course</td>
<td>Normative</td>
</tr>
<tr>
<td>Mode of Studies</td>
<td>full-time</td>
</tr>
<tr>
<td>Year of studies, semester</td>
<td>2 year (3, 4 semester)</td>
</tr>
<tr>
<td>ECTS workload</td>
<td>8 credits (ECTS). Time allotment – 240 hours, including 108 hours of classroom work, and 132 hours of self-study.</td>
</tr>
<tr>
<td>Testing and assessment</td>
<td>4 semester – Final test, 3 semester – Exam,</td>
</tr>
<tr>
<td>Course Schedule</td>
<td>2 classes per week by the timetable <a href="http://rozklad.kpi.ua/">http://rozklad.kpi.ua/</a></td>
</tr>
<tr>
<td>Language of Instruction</td>
<td>English</td>
</tr>
</tbody>
</table>

Course Instructors

- Lecturer: PhD, Associate Professor, Andrii Antoniuk, mobile +380503588019, email antoniuk.andrii@ill.kpi.ua, personal web page [http://se-111.blogspot.com](http://se-111.blogspot.com)
- Teacher of practical work: PhD, Associate Professor, Andrii Antoniuk, mobile +380503588019, email antoniuk.andrii@ill.kpi.ua, personal web page [http://se-111.blogspot.com](http://se-111.blogspot.com)
- Teacher of laboratory work: PhD, Associate Professor, Andrii Antoniuk, mobile +380503588019, email antoniuk.andrii@ill.kpi.ua, personal web page [http://se-111.blogspot.com](http://se-111.blogspot.com)
- Example: PhD, Associate Professor, Alexandr Petrov, mobile +38XXXX, email XXX@XXX, personal web page [http://XXXXX](http://XXXXX)

Access to the course

- Посилання на дистанційний ресурс (Moodle, Google classroom, тощо)

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 3.3. 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 3.8 Generating design templates Prototype, Singleton, Factory Method.
- Section 4. User interface design:
  - Topic 4.1. Purpose and types of user interfaces.
  - 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 5.5. Structural modeling.
- Topic 5.6. Functional modeling.
- Topic 5.7. Modeling of data flows.

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 6.5. Code optimization and refactoring.
- Topic 6.7. Testing process automation tools (JUnit, JMeter).

Section 7. Software project management:
- Topic 7.2. Project content and quality management.
- 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:

Additional literature:
5. Methodology

Lectures

<table>
<thead>
<tr>
<th>№ lectures</th>
<th>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)</th>
</tr>
</thead>
<tbody>
<tr>
<td>№ lectures</td>
<td>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)</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>7.</td>
<td>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</td>
</tr>
<tr>
<td>8.</td>
<td>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</td>
</tr>
<tr>
<td>№</td>
<td>Lectures</td>
</tr>
<tr>
<td>---</td>
<td>----------</td>
</tr>
<tr>
<td>15.</td>
<td>Singleton and Builder patterns.</td>
</tr>
</tbody>
</table>
### 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)

### Behavioral modeling

- Diagrams of states, activities, interactions, sequences, time.
- Structural modeling. Functional modeling. [6, p. 506-537]
- IWS: Simulation using UML

### Modeling of data flows

- IWS: IBM modeling automation tools

### 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

### Ongoing software integration

- IWS: Installation and use of a continuous integration server

### Project management

- Project management tasks. Triangle of constraints. Project content and quality management. [6, p. 562-597]
- IWS: Means of automation of project management

### 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

### 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.

<table>
<thead>
<tr>
<th>№ laboratory work</th>
<th>Name of laboratory work</th>
<th>Number of hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Draft software application. Build a project with Ant. (Section 2, Topics 4)</td>
<td>4</td>
</tr>
</tbody>
</table>
2.1-2.2)  
2.  UML graphic notation. Documentation using JavaDoc. (Section 2, Topics 2.3-2.4)  
4  
(Chapter 3, Topic 3.3)  
4  
(Chapter 3, Topic 3.4)  
4  
(Chapter 3, Topic 3.5)  
4  
(Section 3, Topic 3.6)  
4  
(Section 3, Topic 3.7)  
4  
8.  Creational patterns. Prototype, Singleton, Factory Method. (Section 3, Topic 3.8)  
4  
4  

Total: 36

6. Self-study

<table>
<thead>
<tr>
<th>№ in order</th>
<th>The name of the topic that is submitted for independent study (preparation for classroom classes)</th>
<th>Number of hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>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]</td>
<td>3</td>
</tr>
<tr>
<td>5.</td>
<td>GRASP design templates [Additional 2 p. 105-134; Additional 9 p. 299-386; <a href="https://uk.wikipedia.org/wiki/GRASP">https://uk.wikipedia.org/wiki/GRASP</a>]</td>
<td>1</td>
</tr>
<tr>
<td>7.</td>
<td>Proxy and Flyweight implementations in Java SDK classes [1, p. 191-212]</td>
<td>1</td>
</tr>
<tr>
<td>8.</td>
<td>Implementations of Adapter, Bridge and Facade in Java SDK classes [1, p. 141-161, 183-190]</td>
<td>1</td>
</tr>
<tr>
<td>9.</td>
<td>Implementations of Iterator and Mediator in Java SDK classes [1, p. 249-271]</td>
<td>1</td>
</tr>
<tr>
<td>10.</td>
<td>Implementations of Observer and Strategy in Java SDK classes [1, p. 280-290, 300-308]</td>
<td>1</td>
</tr>
<tr>
<td>11.</td>
<td>Implementations of Command and Visitor in Java SDK classes [1, p. 227-235, 314-327]</td>
<td>1</td>
</tr>
<tr>
<td>12.</td>
<td>Implementations of State and Memento in Java SDK classes [1, p. 272-279, 291-299]</td>
<td>1</td>
</tr>
<tr>
<td>14.</td>
<td>Implementations of Prototype, Factory Method and Abstract Factory in</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td>15.</td>
<td>Singleton and Builder implementations in Java SDK classes [1, p. 102-110, 130-137]</td>
<td></td>
</tr>
<tr>
<td>19.</td>
<td>Simulation using UML [Additional 2, p. 599-602; Additional 5, p. 34-197; Additional 7, 204-246]</td>
<td></td>
</tr>
</tbody>
</table>

### 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](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).

\[ R_s = R_1 + R_2 + R_3 = 100 \text{ points} \]

Each laboratory work consists of 4 parts: protocol (2 points), printout of the program or file with the text of the raw code (2 points), documentation of the program by means of IDE (2 points), protection of laboratory work (2 points), 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:

\[ 9 \text{ laboratory works} \times 8 \text{ points} = 72 \text{ points} \]
\[ 2 \text{ modular control works} \times 4 \text{ points} = 8 \text{ points} \]
\[ \text{Exam} = 20 \text{ points} \]

4-й semester

For modular control tests (R4) and the final test (R5).

\[ R_s = R_4 + R_5 = 100 \text{ points} \]

In the 4th semester 2 modular tests of 30 points each. For the final test the student can receive up to 40 points. As a result, the maximum average weight score is equal to:

\[ 2 \text{ For modular control tests} \times 30 \text{ points} = 60 \text{ points} \]
\[ \text{Final test} = 40 \text{ points} \]

In the 4th semester there is also a course work, which consists of 4 sections: design methods (R6), software application design (R7); software product development (R8) and software product testing (R9). For the readiness of each section the student receives 20 points. For the defense of the course work the student can receive up to 20 points (R10).

\[ R_s = R_6 + R_7 + R_8 + R_9 + R_{10} = 100 \text{ points} \]

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

\[ 4 \text{ розділи} \times 20 \text{ points} = 80 \text{ points} \]
\[ \text{Final defense test} = 20 \text{ points} \]

According to the university regulations on the monitoring of students’ academic progress [https://kpi.ua/document_control](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:

<table>
<thead>
<tr>
<th>Score</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>100-95</td>
<td>Excellent</td>
</tr>
<tr>
<td>94-85</td>
<td>Very good</td>
</tr>
</tbody>
</table>
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:
  2. What conditions must be met for the effective use of Flyweight.
  3. Implement the Builder pattern.
  5. Adapter pattern.
  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.
  11. Bridge pattern.
  12. Transparent Composite Patterns.
  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.
  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.
  25. The main processes of software life cycle.
  26. Proxy pattern.
  27. Strategy pattern for sorting algorithms.
  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.
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.
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.
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.
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.
92. GRASP patterns. Brief description of each pattern.
93. Pattern Builder. How to ensure the existence of only one instance of each particular builder.
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 № 7, 20 June 2021)
Approved by the Faculty Board of Methodology (protocol № 7, 20 June 2021)