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

Кафедра програмного забезпечення комп'ютерних систем

SOFTWARE ENGINEERING METHODOLOGY

Working program of the academic discipline (Syllabus)

Course Requisites	
Cycle of Higher Education	***Second (master)***
Study area	*12 Information technologies*
Speciality	*121 Software engineering*
Educational program	*Software Engineering of Multimedia and Information Retrieval Systems*
Discipline status	*Normative*
Study form	*Daytime*
Year of study, semester	*1 year of training, 2 semester*
Discipline volume	*Lectures: 36 hours, computer workshop: 18 hours, self work: 66 hours.*
Semester control/ control measures	*Final test, modular control work, calendar control*
Course schedule	*According to the schedule for the spring semester of the current academic year (rozklad.kpi.ua)*
Language	*Ukrainian*
Information about head of the course / teachers	*Lecturer: PhD, senior lecturer, Iana Khitsko, iana.khitsko@gmail.com* Computer workshop: *PhD, senior lecturer, Iana Khitsko, iana.khitsko@gmail.com*
Access to course	Google classroom: https://classroom.google.com/c/NTU0ODU5MjM1NzY2?cjc=e7ug4pz

Outline of the Course

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

The purpose *of studying the discipline "Software Engineering Methodology" is the formation of students' abilities to:*

- *analyze requirements for software systems and their design conditions;*
- *choose software systems development methodology in accordance with defined requirements and software design and construction environment;*
- *determine and analyze software quality metrics;*
- *ensure quality inspection of software development artifacts;*
- *provide unit and integration software testing;*
- *determine and analyze software quality metrics;*
- *ensure high-quality refactoring of the existing software code.*

The subject *of the discipline "Software Engineering Methodology" is the mathematical and algorithmic support of the processes of analysis, design, source code construction and refactoring.*

The study of the discipline "Software Engineering Methodology" contributes to the formation of students of general (SK) and professional (FC) competencies necessary for solving practical tasks of professional activity related to the development, improvement and operation of software systems of various purposes:

GC01 *- ability to abstract thinking, analysis and synthesis;*
PC01 *- ability to analyze subject areas, form, classify software requirements;*
PC03 *- ability to design software architecture, model the operation of individual subsystems and modules;*

***PC05** - ability to develop, analyze and apply specifications, standards, rules and guidelines in the field of software engineering;*
***PC06** - ability to effectively manage financial, human, technical and other project resources in the field of software engineering;*
***PC07 -** ability to critically comprehend problems in the field of information technology and at the frontiers of knowledge, to integrate relevant knowledge and solve complex problems in broad or multidisciplinary contexts;*
***PC08** - ability to develop and coordinate processes, stages and iterations of the software life cycle based on the application of modern models, methods and technologies of software development;*
***PC09** - ability to ensure software quality;*
***PC17** - ability to apply software engineering methodologies in practice.*

*Studying the discipline "Software Engineering Methodology" contributes to students' formation of the following **program learning outcomes (PLO)** according to the educational program:*
***PLO01** - know and apply modern professional standards and regulations on software engineering;*
***PLO02** - evaluate and choose effective methods and models of software development, implementation, support and relevant processes management at all stages of the life cycle;*
***PLO03** - build and research models of information processes in the application field;*
***PLO04** - identify information needs and classify data for software design;*
***PLO05** - develop, analyze, justify and systematize software requirements;*
***PLO06** - develop and evaluate software design strategies; substantiate, analyze and evaluate options for design solutions in terms of the final software product quality, resource constraints and other factors;*
***PLO07** - analyze, evaluate and apply at the system level modern software and hardware platforms to solve complex problems of software engineering;*
***PLO08** - develop and modify software architecture to meet customer requirements;*
***PLO09** - choose reasonable paradigms and programming languages for software development; apply modern software development tools in practice;*
***PLO10** - modify existing and develop new algorithmic solutions for detailed software design;*
***PLO11** - ensure quality at all stages of the software life cycle, including the use of relevant models and assessment methods, as well as automated software testing and verification tools;*
***PLO13** - configure software, manage its changes and develop software documentation at all stages of the life cycle;*
***PLO14** - predict the development of software systems and information technology;*
***PLO15** - carry out software reengineering in accordance with customer requirements;*
***PLO16** - plan, organize and perform software testing, verification and validation;*
***PLO17** - collect, analyze, evaluate the information needed to solve scientific and applied problems, using scientific and technical literature, databases and other sources;*
***PLO21 -** know the theoretical foundations underlying research methods of information systems and software, research methodologies and computational experiments.*

2. Discipline prerequisites and postrequisites (place in the structural and logical education scheme according to the relevant educational program)

The successful study of the discipline "Software Engineering Methodology" is preceded by the study of the disciplines "Programming", "Object-Oriented Programming", "Software Quality", "Software Requirements" of the curriculum for bachelor's training in the specialty 121 Software Engineering.

The theoretical knowledge and practical skills obtained during the mastering of the discipline "Software Engineering Methodology" ensure the successful completion of course projects and master's theses in the specialty 121 Software Engineering.

3. Content of the course

The discipline «"Software Engineering Methodology» involves the study of such topics:

Topic 1. Software engineering methodologies

Topic 2. Software metrics

Modular testw 1

Topic 3. Software verification

Topic 4. Refactoring

Modular test 2

Test

4. Coursebooks and teaching resources

Basis reference:

1. *Electronic campus of NTUU "KPI by Igor Sikorsky". Discipline materials for "Software engineering methodology". – http://login.kpi.ua*
2. *Web-portal of Applied Mathematics Faculty. Materials archive. «Хіцко» folder. – Режим доступу : http://fpm.kpi.ua/archive/dir.do?sys_id=obj_2*

Additional reference:

3. *Project Management Institute, Inc. / A Guide to the Project Management Body of Knowledge: PMBOK Guide. Fifth Edition. — PMI, 2013. — 589 pp. — ISBN-13: 860-1200917796.*
4. *Booch, G., Rumbaugh, J., Jacobcon, I. The Unified Modeling Language User Guide [Text] / G. Booch, J. Rumbaugh, I. Jacobcon — Addison-Wesley, 1998. — 512 p. — ISBN 0-201-57168-4.*
5. *Kent Beck. "Manifesto for Agile Software Development" [електронний ресурс] / Kent Beck, James Grenning, Robert C. Martin, Mike Beedle, Jim Highsmith, Steve Mellor, Arie van Bennekum, Andrew Hunt, Ken Schwaber, Alistair Cockburn, Ron Jeffries, Jeff Sutherland, Ward Cunningham, Jon Kern, Dave Thomas, Martin Fowler, Brian Marick // Agile Alliance, 2001. – режим доступу - http://agilemanifesto.org/ .*
6. *Steve McConnell (1996). Rapid Development: Taming Wild Software Schedules, Microsoft Press Books, ISBN 978-1-55615-900-8*
7. *DSDM Consortium. DSDM Atern: the Handbook [електронний ресурс] / DSDM Consortium, 2008. – режим доступу - https://www.agilebusiness.org/resources/dsdm-handbooks/dsdm-atern-handbook-2008 .*
8. *Schwaber, Ken (2004). "SCRUM Development Process"(PDF). Advanced Development Methods - режим доступу - http://www.jeffsutherland.org/oopsla/schwapub.pdf*
9. *"Extreme Programming Rules". extremeprogramming.org.*
10. *Palmer, S.R., & Felsing, J.M. (2002). A Practical Guide to Feature-Driven Development. Prentice Hall. (ISBN 0-13-067615-2)*
11. *Cockburn, Alistair. Crystal Clear, A Human-Powered Methodology for Small Teams [Text] / Alistair Cockburn // Addison-Wesley Professional, 2004. – pp.336. - ISBN 0-201-69947-8.*
12. *Kanban: Successful Evolutionary Change for Your Technology Business, David J. Anderson. (United States, Blue Hole Press, 2010. ISBN 978-0984521401*
13. *Scrumban: Essays on Kanban Systems for Lean Software Development, Corey Ladas. (United States, Modus Cooperandi Press, 2009. ISBN 9780578002149*
14. *Mary Poppendieck; Tom Poppendieck (2003). Lean Software Development: An Agile Toolkit. Addison-Wesley Professional. ISBN 978-0-321-15078-3.*
15. *Cohn, Mike. "Planning Poker Cards: Effective Agile Planning and Estimation". Mountain Goat Software, 30 March 2016. - режим доступу - https://www.mountaingoatsoftware.com/tools/planning-poker*
16. *Eric Evans, 2015 Domain Driven Design, Definitions and Pattern Summaries. Режим доступу - https://domainlanguage.com/wp-content/uploads/2016/05/DDD_Reference_2015-03.pdf*

17. *Крачтен Филипп. Введение в Rational Unified Process / Крачтен Филипп // 2-е изд.: Пер. с англ. — М.: Издательский дом "Вильямс", 2002. — 240 с.*
18. *Halstead, M. H. Elements of Software Science [Text] / M. H. Halstead. — New York, Elsevier North-Hollan, 1977. — p. 128. — ISBN 0444002057.*
19. *Chidamber, S. R. A Metrics Suite for Object Oriented Design [Text] / S.R. Chidamber, C. F. Kemerer // IEEE Transactions on Software Engineering. — June 1994. — vol. 20, No. 6. — pp. 476-493.*
20. *Graham, I. Object-Oriented Methods. Principles & Practice [Text] / I. Graham. 3rd Edition. — Addison-Wesley, 2000. — 864 pp. — ISBN 978-0201619133.*
21. *Hitz, M. Measuring Coupling in Object-Oriented Systems. [Text] / M. Hitz, B. Montazeri // Object Currents. — Apr 1996. — vol. 2, No. 4. — 17 pp.*
22. *Lorenz, M. Object-Oriented Software Metrics [Text] / M. Lorenz, J. Kidd. — Prentice Hall, 1994. — 146 p. — ISBN 978-0131792920.*
23. *Humphrey, Watts. The Personal Unified Process [Text] /* Humphrey, W. S. // Mass.: Addison-Wesley, 1983. - ISBN 0-201-18095-2.
24. *Рефакторинг. Поліпшення існуючого коду / М. Фаулер, К. Бек, Дж. Брант, В. Опдайк, Д. Робертс. – «Діалектика», 2003. – 448 с.*

Educational content

5. Methods of mastering the discipline (educational component)

№	Training session type	Lesson description
	Topic 1. Software development life cycle	
1	*Lecture 1. Software development life cycle*	*Phases of the project according to the PMI methodology and software development process. Life cycle models. Waterfall model. Prototyping model. The big bang model. V-shaped. Iteration. Incremental and iterative models. Spiral model.*
2	*Lecture 2. Software development methodologies*	*Basic software development methodologies. RAD. Basic principles of Agile. Basic concepts and practices of XP. XP development cycle.*
3	*Lecture 3. Scrum*	*Scrum: basic concepts and development cycle. Project backlog, user stories and meetings in Scrum.*
4	*Computer Workshop 1. SCRUM*	*Objective: plan a project and develop three iterations using the SCRUM process.*
5	*Lecture 4. Methodologies DSDM, FDD and Crystal, KANBAN*	*Basic concepts and practices of DSDM, FDD, KANBAN and Crystal methodologies.*
6	*Lecture 5. Agile practices*	*Iterative and incremental development. Sprint and project backlog. User stories and their estimation methods. Planning pocker and velocity in Agile.*
7	*Lecture 6. RUP*	*Rational Unified Process: dynamic and static aspects, disciplines, phases, iterations and artifacts.*
8	*Lecture 7. Software configuration processes*	*Configuration Management Process. Release management process.*
	Topic 2. Software metrics	

9	*Lecture 8. Software metrics. Quantitative metrics*	*Software metrics. Types of metrics. Application size metrics. LOC assessments. Metrics of the level of commenting of the software code. Halstead metrics. Functionally oriented metrics. Number of functional points.*
10	*Lecture 9. Complexity metrics*	*Program control flow complexity metrics. McCabe Cyclomatic Complexity Metrics. Program control flow complexity metrics. Data flow complexity metrics. The 'module - global variable' metric. Chepino metric. Metrics of style and comprehensibility of programs. Metrics of the level of commenting of the software code.*
11	*Computer workshop 2. Quantitative software metrics*	*Objective: to develop a program that calculates quantitative metrics of arbitrarily chosen program code in any programming language*
12	*Lecture 10. Features of software structuring*	*Peculiarities of system structuring. Module cohesion. Types of cohesion. Module coupling.*
13	*Lecture 11. Object-oriented metrics*	*Object-oriented metrics. Objects coupling. Coupling metrics by methods. Dependence of changes between classes - class level coupling.*
14	*Lecture 12. Object-oriented metrics*	*Chidamber and Kemerer metrics - WMC, DIT, NOC, CBO, RFC-LCOM. Lorentz and Kidd metrics. Code support metrics.*
15	*Computer workshop 3. Object-oriented hmetrics*	*Objective: develop a program that calculates object-oriented metrics of any program code in any programming language*
Modular test 1		
Topic 3. Software verification		
16	*Lecture 13. Software verification and validation*	*Differences between software verification and validation. Artifacts of a software development project. Verification of the design solution, plan, requirements, test plans. Software audit. Expertise. Methods of static analysis. Dynamic methods*
17	*Lecture 14. Code reviews and code inspections*	*Basic concepts of software technical review. Basic aspects of technical review metrics analysis. Formal code inspection - process, process artifacts, and its participants. Continuous review, purpose and roles. Congitive continuous check*
18	*Computer workshop 4. Code inspections*	*Objective: develop a software utility, make its code freely available to enable code inspection and further improvement.*
Topic 4. Refactoring		
19	*Lecture 15. Introduction to refactoring. Code smells.*	*What is refactoring and when to do it. What are code smells. Classification of source code problems. Violation of object-oriented design. Violation of norms of volumes of source*

		code. Complicators of code changes. Source code contaminants, confusing object relationships, library incompleteness.
20	*Lecture 16. Refactoring at the level of data and operators. Compilation of methods.*	*Refactoring methods at the level of data organization and conditional operators, method compilation.*
21	*Lecture 17. Simplifying method calls. Moving functions between objects.*	*Refactoring methods: simplifying method calls and moving functions between objects.*
22	*Lecture 18. Solving generalization problems. Refactoring at the system level.*	*Refactoring methods: solving generalization problems. Separation of imitation. Converting procedural code into objects. Separation of the subject area from the presentation. Highlighting the hierarchy.*
Modular test 2		

6. Self-study

The discipline "Software Engineering Methodology" is based on independent preparations for classroom classes on theoretical and practical topics.

№	*The name of the topic that is submitted for independent study*	*Hours of study*	*References*
1	*Preparing for lecture 1*	*1*	*1, pp. 9-28; 3, pp. 8-10; 4, pp. 8-18*
2	*Preparing for lecture 2*	*1*	*3, pp. 54-63; 4, pp. 42-43, 47-49; 5, pp. 32-41;*
3	*Preparing for lecture 3*	*1*	*5, 7*
4	*Preparing for computer workshop 1*	*2*	*8*
5	*Preparing for lecture 4*	*1*	*11, pp. 44-47; 12-13*
6	*Preparing for lecture 5*	*1*	*5-6, 14-16*
7	*Preparing for lecture 6*	*1*	*17, pp. 15-30*
8	*Preparing for lecture 7*	*1*	*18*
9	*Preparing for lecture 8*	*1*	*19*
10	*Preparing for lecture 9*	*1*	*18-19*
11	*Preparing for computer workshop 2*	*2*	*18-19*
12	*Preparing for lecture 10*	*1*	*20, pp. 20-24; 21, pp. 41-48*
13	*Preparing for lecture 11*	*1*	*20, pp. 20-24; 21, pp. 41-48*
14	*Preparing for lecture 12*	*1*	*22, pp. 11-18*
15	*Preparing for computer workshop 3*	*2*	*20, pp. 20-24; 21, pp. 41-48; 22, pp. 11-18*
16	*Preparing for modular test 1*	*6*	*5-8; 11, pp. 44-47; 12-13; 17, pp. 15-30;*

			18-19; 20, pp. 20-24; 21, pp. 41-48; 22, pp. 11-18
17	*Preparing for lecture 13*	*1*	*23, pp. 226-234*
18	*Preparing for lecture 14*	*1*	*23, pp. 226-234*
19	*Preparing for computer workshop 4*	*2*	*23, pp. 226-234*
20	*Preparing for lecture 15*	*1*	*24, pp. 15-34*
21	*Preparing for lecture 16*	*1*	*24, pp. 35-48*
22	*Preparing for lecture 17*	*1*	*24, pp. 50-64*
23	*Preparing for lecture 18*	*1*	*24, pp. 72-86*
24	*Preparing for modular test 2*	*6*	*23, pp. 226-234; 24, pp. 15-86*
25	*Preparing for final test*	*8*	*5-8; 11, pp. 44-47; 12-13; 17, pp. 15-30; 18-19; 20, pp. 20-24; 21, pp. 41-48; 22, pp. 11-18; 23, pp. 226-234; 24, pp. 15-86*
26	*Main practices of Test driven design.*	*4*	*4r, pp. 6-8*
27	*Integration management processes. Server setup for early integration and release management skills.*	*4*	*6-7*
28	*Test driven development. To implement the task for laboratory work 5 using TDD.*	*4*	*4, pp. 16-18*
29	*Coupling and cohesion metrics. Analysis of an arbitrary software library for connectivity and coupling metrics using the Visual Studio development environment.*	*4*	*22, pp. 11-18*
30	*Refactoring: define a refactoring method for a given software module.*	*4*	*20, pp. 20-24*

Policy and Assessment

7. Course policy

- *• Attending lectures is mandatory.*
- *• Attending computer workshop classes may be occasional and as needed to protect computer workshop work.*
- *• Rules of behavior in classes: activity, respect for those present, turning off phones.*
- *• Adherence to the policy of academic integrity.*
- *• Rules for protecting the works of the computer workshop: the works must be done according to the option of the student, which is determined by his number in the group list.*
- *• The rules for assigning incentive and penalty points are as follows..*

Penalty points are calculated for:

- plagiarism (the program code does not correspond to the task variant, the identity of the program code among different works) in the works of the computer workshop: -5 points for each attempt.

8. Types of control and rating system for evaluating learning outcomes (ELO)

During the semester, students complete 4 computer workshops. The maximum number of points for each computer workshop: 15 points.

Points are awarded for:
- quality of computer workshop: 0-7 points;
- answer during the defense of computer workshop: 0-4 points;
- timely submission of work for defense: 0-4 points.

Criteria for evaluating the quality of performance:
6-7 points - the work is done qualitatively, in full;
4-5 points - the work is done qualitatively, in full, but has shortcomings;
1-3 points – the work is completed in full, but contains minor errors;
0 points – the work is incomplete or contains significant errors.

Answer evaluation criteria:
4 points – the answer is complete, well-argued;
3 points – the answer is complete, but not sufficiently well argued;
2 points – the answer is generally correct, but has flaws or minor errors;
1 point – there are significant errors in the answer;
0 points - there is no answer or the answer is incorrect.

Criteria for evaluating the timeliness of work submission for defense:
4 points – the work is presented for defense no later than the specified deadline;
3 points – the work is submitted for defense later than the specified deadline (up to one week);
2 points – the work is submitted for defense later than the specified deadline (from one to two weeks);
1 point – the work is submitted for defense later than the specified period (from two to three weeks);
0 points – the work is submitted for defense later than the specified deadline (from three weeks).

The maximum number of points for computer workshop:
15 points × 4 workshops = 60 points.

During the semester, students complete two modular tests. The assignment for the ***modular test*** *consists of 5 questions - 4 theoretical and 1 practical. The answer to each question is evaluated by 4 points.*

Evaluation criteria for each theoretical test question:
4 points – the answer is correct, complete, well-argued;
3 points - in general, the answer is correct, but has flaws;
2 points – there are minor errors in the answer;
1 point – there are significant errors in the answer;
0 points - there is no answer or the answer is incorrect.

Evaluation criteria for the practical test question:
4 points – the answer is correct, the calculations are complete;
3 points - in general, the answer is correct, but has flaws;
2 points – there are minor errors in the answer;
1 point – there are significant errors in the answer;
0 points - there is no answer or the answer is incorrect.

The maximum number of points for a modular control work:
4 points × 4 theoretical questions + 4 points × 1 practical question = 20 points.

The maximum number of points for modular control works:
20 points × 2 tests = 40 points.

The rating scale for the discipline is equal to:

R = RS = 60 points + 40 points = 100 points.

Calendar control: is carried out twice a semester as a monitoring of the current state of fulfillment of the syllabus requirements. At the first certification (8th week), the student receives "passed" if his current rating is at least 12 points (50% of the maximum number of points a student can receive before the first certification). At the second certification (14th week), the student receives "passed" if his current rating is at least 20 points (50% of the maximum number of points a student can receive before the second certification).

Semester control: final test
Conditions for admission to semester control:
With a semester rating (RC) of not less than 60 points and the enrollment of all computer workshop, the student receives passing the test "automatically" according to the table (Table of correspondence of rating points to grades on the university scale). Otherwise, he/she has to perform the final test.

Completion and defense of a computer workshop is a necessary condition for admission to the final test.

If the student does not agree with the "automatic" grade, he/she can try to improve his/her grade by writing a final test, while his points received for the semester are kept, and the better of the two grades received by the student is assigned ("soft" grading system).

Points	*Grade*
100-95	Excellent
94-85	Very good
84-75	Good
74-65	Satisfactorily
64-60	Enough
< 60	Unsatisfactorily
Admission conditions are not met	Not admitted

9. Additional information about the course

The list of questions to be submitted for semester control is given in Appendix 1.

Course syllabus:

Is created by PhD, senior lecturer Iana Khitsko**.**

Adopted by Computer Systems Software Department (protocol № 12 from 26.04.23)

Approved by the Faculty Board of Methodology (protocol № 10 from 26.05.23)

Appendix 1. List of questions to be submitted for semester control

1. Name the main practices in Feature Driven Design (minimum 3).

2. How does a daily Scrum meeting go and what issues are discussed?

3. Are intermediate product releases distributed in cascade and incremental software development models?

4. What scale of projects can use Crystal Orange?

5. What are the conditions for applying the RAD methodology?

6. Is there a sprint finish in Scrumban?

7. Is there a sprint finish in Kanban?

8. How does a user story differ from a use case?

9. How is team performance evaluated in Scrum?

10. What is the anti-pattern 'watch the master' in pair programming?

11. How can requirements change in cascade and incremental software development models?

12. Can an architect or a project manager pre-determine task estimates in the planning poker process?

13. What are story points used for?

14. What is the MoSCoW principle?

15. What are mock and fake objects?

16. What is the difference between a technical software review and a formal inspection?

17. Which parts of the project and how will the reduction of the project budget affect?

18. Is it necessary to rewrite drivers at each stage of downward integration testing?

19. In what phase of RUP is the system evaluation process?