ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ЗАТВЕРДЖУЮ
Декан факультету
інформаційних технологій
_____ Олег САВЕНКО
_____ 2023 р.

РОБОЧА ПРОГРАМА НАВЧАЛЬНОЇ ДИСЦИПЛІНИ
Конструювання програмного забезпечення

Галузь знань 12 – Інформаційні технології
Спеціальність 121 – Інженерія програмного забезпечення
Рівень вищої освіти – Перший бакалаврський
Освітньо-професійна програма – Інженерія програмного забезпечення
Обсяг дисципліни –6 кредитів ЄКТС, ***Шифр дисципліни*** – ОПП.08
Статус дисципліни: обов'язкова, ***Мова навчання*** Англійська, українська
Факультет – Інформаційних технологій
Кафедра – Інженерії програмного забезпечення

Форма навчання	Курс	Семестр	Загальне навантаження		Кількість годин						Курсовий проект	Курсова робота	Форма семестрового контролю	
					Аудиторні заняття				Індивідуальна робота студента	Самостійна робота студента в т.ч. ІРС				
			Кредити ЄКТС	Години	Разом	Лекції	Лабораторні роботи	Практичні заняття					Залік	Іспит
Очна (денна)	4	7	6	180	68	34	34			112				+
Разом			**6**	**180**	**68**	**34**	**34**			**112**				**1**

Робоча програма складена на основі Стандарту вищої освіти, освітньої програми підготовки бакалаврів 2023 року та навчального плану.

Програма складена Наталія ПРАВОРСЬКА

Схвалено на засіданні кафедри ІПЗ
протокол № 1 від 31 08 2023 р.
Зав. кафедри інженерії програмного забезпечення Леонід БЕДРАТЮК

Робоча програма розглянута та схвалена Вченою радою факультету інформаційних технологій

Голова Вченої ради факультету Олег САВЕНКО

Хмельницький 2023

КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Тип дисципліни	Обов'язкова
Рівень вищої освіти	Перший
Мова викладання	Англійська, українська
Семестр	Сьомий
Обсяг кредитів ЄКТС	6
Форма здобуття освіти	Очна(денна)

Результати навчання. Студент, який успішно завершив вивчення дисципліни, має: аналізувати, цілеспрямовано шукати і вибирати необхідні для вирішення професійних завдань інформаційно-довідникові ресурси і знання з урахуванням сучасних досягнень науки і техніки. Уміння вибирати та використовувати відповідну задачі методологію створення програмного забезпечення. Вміти розробляти людино-машинний інтерфейс. Проводити передпроєктне обстеження предметної області, системний аналіз об'єкта проектування. Вибирати вихідні дані для проектування, керуючись формальними методами опису вимог та моделювання. Застосовувати на практиці ефективні підходи щодо проектування програмного забезпечення. Знати і застосовувати методи розробки алгоритмів, конструювання програмного забезпечення та структур даних і знань. Застосовувати на практиці інструментальні програмні засоби доменного аналізу, проектування, тестування, візуалізації, вимірювань та документування програмного забезпечення. Мати навички командної розробки, погодження, оформлення і випуску всіх видів програмної документації. Вміти застосовувати методи компонентної розробки програмного забезпечення. Знати та вміти застосовувати методи верифікації та валідації програмного забезпечення. Вміти ефективно співпрацювати з виконавцями різних етапів життєвого циклу програмного забезпечення, координуючи одночасну роботу над етапами. Знати та вміти застосовувати методики, інструменти та стратегії управління ресурсами, часом та комунікацією для забезпечення ефективної паралельної роботи над різними етапами життєвого циклу програмного забезпечення.

Пререквізити – Архітектура та проєктування програмного забезпечення, Моделювання та оцінка програмного забезпечення

Кореквізити – Професійна практика

Зміст навчальної дисципліни. Елементи конструювання ПЗ. Ключові принципи конструювання. Інструменти конструювання. Мови опису, представлення, архітектурні каркаси. Техніка моделювання архітектури. Ідентифікація цілей та ключових сценаріїв. Системний підхід до розробки ПЗ. Моделі життєвого циклу ПЗ. Методологія створення ПЗ. Ідентифікація ключових проблем. Конструювання ПЗ та багатозадачність. Захисне конструювання ПЗ, синхронізація процесів, підходи при конструюванні ПЗ. Відладки при конструюванні ПЗ та контроль за ними.

Запланована навчальна діяльність: лекцій 34 год., лабораторних занять 34 год., самостійної роботи 112 год.; разом 180 год.

Методи навчання методи проблемного викладання, словесні, наочні (лекції); пояснювально-ілюстративні, проблемного викладання, дослідницькі, частково-пошукові (лабораторні заняття), проблемного викладання, дослідницькі, частково-пошукові (самостійна робота: індивідуальні завдання).

Форми і методи оцінювання результатів навчання: усне опитування, захист лабораторних робіт. письмові самостійні та контрольні роботи, письмовий іспит

Форма семестрового контролю: іспит

Навчальні ресурси:

1. Роберт Мартін. Чистий код: Чистий код: створення, аналіз, рефакторинг. - Фабула. 2019 – 416 с.
2. Постіл. С.Д. UML. Уніфікована мова моделювання інформаційних систем: Навч. посіб., 2019. - 321 с.
3. Martin Fowler. Refactoring: Improving the Design of Existing Code (WebEdition), 2nd Edition/ WebEdition.- 2018
4. Aho, Alfred V.,Jeffrey D. Ullman. The Theory of Parsing, Translation, and Compiling (Volume 2: Compiling). -Prentice-Hall. -2018
5. Ian Sommerville. Software Engineering, 10th edition. Published by Pearson . July 14th 2021 – 816 p.
6. Модульне середовище для навчання MOODLE. Доступ до ресурсу: https://msn.khmnu.edu.ua

Викладач: кандидат педагогічних наук, доцент Наталя ПРАВОРСЬКА

3. ПОЯСНЮВАЛЬНА ЗАПИСКА

Дисципліна “ «Конструювання програмного забезпечення» є однією із дисциплін професійної підготовки і займає провідне місце у підготовці фахівців освітнього рівня "бакалавр” за освітньо-професійною програмою "Інженерія програмного забезпечення".

Пререквізити – Архітектура та проєктування програмного забезпечення, Моделювання та оцінка програмного забезпечення
Кореквізити – Професійна практика

Мета дисципліни – формування теоретичних знань та практичних навичок у сфері контруювання програмного забезпечення.

Предмет дисципліни: технології проектування, моделі і методи підтримки життєвого циклу програмного забезпечення, засоби та методи створення та реалізації проектів.

Завдання дисципліни: ознайомлення здобувачів з сучасними методами та технологіями конструктивного програмного забезпечення; вивчення способів конструювання програмного забезпечення; формування вмінь та навиків виробітки конструкторських рішень; формування навичок роботи у сучасних інструментальних середовищах підтримки процесу конструювання програмних систем; розвиток умінь організовувати паралельну роботу над етапами життєвого циклу програмного забезпечення для оптимізації процесів розробки; набуття навичок впровадження ефективних комунікаційних стратегій та використання інструментів для забезпечення взаємодії між виконавцями етапів розробки програмного забезпечення.

Відповідно до ***Стандарту вищої освіти*** із та освітньої програми дисципліна сприяє забезпеченню:

компетентностей:

ЗК02. Здатність застосовувати знання у практичних ситуаціях.

ЗК07. Здатність працювати в команді.

ФК02. Здатність брати участь у проєктуванні програмного забезпечення, включаючи проведення моделювання (формальний опис) його структури, поведінки та процесів функціонування.

ФК10. Здатність накопичувати, обробляти та систематизувати професійні знання щодо створення і супроводження програмного забезпечення та визнання важливості навчання протягом всього життя.

ФК11. Здатність реалізовувати фази та ітерації життєвого циклу програмних систем та інформаційних технологій на основі відповідних моделей і підходів розробки програмного забезпечення.

ФК12. Здатність здійснювати процес інтеграції системи, застосовувати стандарти і процедури управління змінами для підтримки цілісності, загальної функціональності і надійності програмного забезпечення.

ФК15. Здатність до паралельної роботи над етапами життєвого циклу програмного забезпечення та ефективної взаємодії між виконавцями етапів.

програмних результатів навчання:

ПРН01. Аналізувати, цілеспрямовано шукати і вибирати необхідні для вирішення професійних завдань інформаційно-довідникові ресурси і знання з урахуванням сучасних досягнень науки і техніки.

ПРН06. Уміння вибирати та використовувати відповідну задачі методологію створення програмного забезпечення.

ПРН08. Вміти розробляти людино-машинний інтерфейс.

ПРН10. Проводити передпроєктне обстеження предметної області, системний аналіз об'єкта проектування.

ПРН11. Вибирати вихідні дані для проектування, керуючись формальними методами опису вимог та моделювання.

ПРН12. Застосовувати на практиці ефективні підходи щодо проектування програмного забезпечення.

ПРН13. Знати і застосовувати методи розробки алгоритмів, конструювання програмного забезпечення та структур даних і знань.

ПРН14. Застосовувати на практиці інструментальні програмні засоби доменного аналізу, проектування, тестування, візуалізації, вимірювань та документування програмного забезпечення.

ПРН16. Мати навички командної розробки, погодження, оформлення і випуску всіх видів програмної документації.

ПРН17. Вміти застосовувати методи компонентної розробки програмного забезпечення.

ПРН19. Знати та вміти застосовувати методи верифікації та валідації програмного забезпечення.

ПРН25 Вміти ефективно співпрацювати з виконавцями різних етапів життєвого циклу програмного забезпечення, координуючи одночасну роботу над етапами

ПРН26 Знати та вміти застосовувати методики, інструменти та стратегії управління ресурсами, часом та комунікацією для забезпечення ефективної паралельної роботи над різними етапами життєвого циклу програмного забезпечення.

Результати навчання. Студент, який успішно завершив вивчення дисципліни, має: аналізувати, цілеспрямовано шукати і вибирати необхідні для вирішення професійних завдань інформаційно-довідникові ресурси і знання з урахуванням сучасних досягнень науки і техніки. Уміння вибирати та використовувати відповідну задачі методологію створення програмного забезпечення. Вміти розробляти людино-машинний інтерфейс. Проводити передпроєктне обстеження предметної області, системний аналіз об'єкта проектування. Вибирати вихідні дані для проектування, керуючись формальними методами опису вимог та моделювання. Застосовувати на практиці ефективні підходи щодо проектування програмного забезпечення. Знати і застосовувати методи розробки алгоритмів, конструювання програмного забезпечення та структур даних і знань. Застосовувати на практиці інструментальні програмні засоби доменного аналізу, проектування, тестування, візуалізації, вимірювань та документування програмного забезпечення. Мати навички командної розробки, погодження, оформлення і випуску всіх видів програмної документації. Вміти застосовувати методи компонентної розробки програмного забезпечення. Знати та вміти застосовувати методи верифікації та валідації програмного забезпечення. Вміти ефективно співпрацювати з виконавцями різних етапів життєвого циклу програмного забезпечення, координуючи одночасну роботу над етапами. Знати та вміти застосовувати методики, інструменти та стратегії управління ресурсами, часом та комунікацією для забезпечення ефективної паралельної роботи над різними етапами життєвого циклу програмного забезпечення.

Політика дисципліни Організація освітнього процесу з дисципліни відповідає вимогам положень про організаційне і навчально-методичне забезпечення освітнього процесу, освітній програмі та навчальному плану. Студент зобов’язаний відвідувати лекції, практичні заняття, лабораторні роботи, тощо, згідно з розкладом, не запізнюватися на заняття, виконувати усі завдання та контрольні точки відповідно до графіка. Пропущені практичні заняття і лабораторні роботи студент зобов’язаний опрацювати самостійно у повному обсязі і відзвітувати перед викладачем не пізніше, ніж за тиждень до чергової атестації. До практичних занять і лабораторних робіт студент має підготуватися за відповідною темою і проявляти активність. Набутті особою знання з дисципліни або її окремих розділів у неформальній освіті зараховуються відповідно до Положення про порядок перезарахування результатів навчання та визначення академічної різниці у ХНУ.

4. СТРУКТУРА ЗАЛІКОВИХ КРЕДИТІВ ДИСЦИПЛІНИ

“Конструювання програмного забезпечення”

Теми	Лекції	Лаб. роб.	Самостійна робота здобувачів
Тема 1. Предмет і зміст дисципліни. Місце дисципліни в структурі навчання. Проблеми розробки ПЗ та шляхи їх вирішення. Інженерія ПЗ	2	2	12
Тема 2. Системний підхід до розробки ПЗ. Моделі життєвого циклу ПЗ. Методологія створення ПЗ	4	4	12
Тема 3. Проектування при конструюванні ПЗ	8	4	12
Тема 4. Конструювання ПЗ та багатозадачність.	2	4	12
Тема 5. Захисне конструювання ПЗ, синхронізація процесів, підходи при конструюванні ПЗ (структурний та ООП)	4	4	12
Тема 6. Евристичні принципи конструювання програм та із захистом від помилок	2	4	12
Тема 7. Методи обробки помилок, аварійний захист ПЗ. Технологія налагодження ПЗ.	4	4	12
Тема 8. Відладки при конструюванні ПЗ та контроль за ними.	6	4	12
Тема 9. Моделювання роботи СТС Успадковане ПЗ. Мобільність ПЗ і реінженіринг ПЗ.	2	4	16
Разом за 8-й семестр	**34**	**34**	**112**

5. ПРОГРАМА НАВЧАЛЬНОЇ ДИСЦИПЛІНИ

“Конструювання програмного забезпечення”

5.1. Зміст лекційного курсу*

№ лекції	Перелік тем лекцій, їх анотація	Кількість годин
Тема 1. Предмет і зміст дисципліни. Місце дисципліни в структурі навчання. Проблеми розробки ПЗ та шляхи їх вирішення. Інженерія ПЗ		2
1	**Лекція 1. Проблеми розробки ПЗ та шляхи їх вирішення. Інженерія ПЗ. Розв’язання професійних задач за допомогою сучасних досягнень науки і техніки.** Програмування заданих прикладів і рішення задач за допомогою ПЗ – в чому різниця? Роль ПЗ і комп’ютерів у виробництві, соціальному житті і науці. Інженерія ПЗ. Проблеми розробки ПЗ та шляхи їх вирішення. Аналіз предметної області та пошук розв’язання задач проєктування за допомогою сучасних досягнень науки і техніки. Передпроєктне обстеження предметної області, системний аналіз об'єкта проектування, вибір необхідних для вирішення професійних завдань інформаційно-довідникові ресурсів з урахуванням сучасних досягнень науки і техніки. Література: [1-3; 7; 12].	2
Тема 2. Системний підхід до розробки ПЗ. Моделі життєвого циклу ПЗ. Методологія створення ПЗ.		4
2	**Лекція2. Якість ПЗ та фактори, які на нього впливають. Методи компонентної розробки програмного забезпечення.** Технологія розробки ПЗ і якість ПЗ. Вибір методології створення програмного забезпечення. Характеристики якості ПЗ, які є важливими для користувача. Фактори, що впливають на якість ПЗ. Література: [2; 5; 13].	2
3	**Лекція 3. Системний підхід до розробки ПЗ. Каскадна модель життєвого циклу ПЗ. Конструювання ПЗ. Критерії якості ПЗ розробника. Стандарти та розробка ПЗ.** Тимчасовий і «просторовий»	2

	аспекти системного підходу Етапи життєвого циклу ПЗ. Каскадна модель життєвого циклу. Конструювання ПЗ і внутрішні критерії якості ПЗ – критерії розробника. Стандарти по розробці ПЗ. Види і значення стандартів, вимоги стандартів. Вибір вихідних даних для проектування на основі формальних методів опису вимог та моделювання. Три групи процесів створення ПЗ. Література: [3-4; 11, 14].	
Тема 3. Проектування при конструюванні ПЗ.		8
4	**Лекція 4. Верифікація ПЗ. V-образна схема життєвого циклу ПЗ. спіральна модель ЖЦ ПЗ. Технології розробки ПЗ: «важкі і легкі», методологія Agile, XP-програмування, DevOps (синхронне виконання етапів ЖЦ).** Життєвий цикл ПЗ і процеси верифікації. Тестування, верифікація, валідація V-образна модель життєвого циклу ПЗ Спіральна модель ЖЦ ПО. «Важкі і швидкі» технології розробки ПЗ. Методологія Agile (робота в команді). Екстремальне (XP) програмування. Технології DevOps (інструменти та стратегії управління ресурсами, часом та комунікацією – синхронне виконання етапів ЖЦ) . Література: [4; 7; 8-10, 15].	2
5	**Лекція 5. Паралельна робота над етапами життєвого циклу програмного забезпечення.** Поняття та важливість паралельної роботи у життєвому циклі програмного забезпечення. Переваги та виклики паралельного виконання робіт на різних етапах розробки. Методи та інструменти для координації паралельної роботи, включаючи Agile та Scrum підходи. Техніки ефективної взаємодії між командами на різних етапах життєвого циклу ПЗ, включаючи планування, розробку, тестування та впровадження. Роль DevOps у підтримці паралельної роботи та забезпеченні неперервної інтеграції/неперервного розгортання (CI/CD). Практичні приклади застосування паралельної роботи для скорочення часу розробки та підвищення якості ПЗ. . Література: [2; 4-6; 16].	2
6	**Лекція 6. Комунікаційні стратегії та інструменти для ефективної взаємодії між виконавцями етапів розробки ПЗ**. Огляд основних комунікаційних моделей та їх застосування у контексті розробки програмного забезпечення. Важливість зворотного зв'язку та його роль у паралельній роботі. Технології співпраці та комунікації, такі як системи управління проектами, чати, відеоконференції та інші інструменти спільної роботи. Виклики та кращі практики управління конфліктами та непорозуміннями між командами. Методи побудови ефективних міжкомандних відносин і культури співпраці у багатодисциплінарних проєктах. Приклади успішної взаємодії між командами на різних етапах життєвого циклу ПЗ. Література: [1-3; 7; 16]	2
7	**Лекція 7. Структурна схема ПЗ. СТС, як засіб конструювання ПЗ. Ієрархічна структура ПЗ СТС. Циклічність (періодичність) у часі вирішення завдань ПЗ. Співпраця між виконавцями різних етапів ЖЦ ПЗ.** Структура системи, ієрархія управління і структура ПЗ. Розвиток системи і проведення змін в ієрархічній структурі ПЗ. Циклічність (періодичність) у часі вирішення завдань управління і роботи ПЗ. Співпраця команд розробки та команд операцій на різних етапах ЖЦ при розробці ПЗ. Література: [4-5; 6]	2
Тема 4. Конструювання ПЗ та багатозадачність.		2
8	**Лекція 8. Тимчасова діаграма роботи системи, як засіб конструювання ПЗ, яке забезпечує паралельні фізичні процеси. Багатозадачна робота ПЗ і її причини.** Тимчасова діаграма роботи системи і ПЗ з паралельними фізичними процесами. Проблема поділу ресурсів для безпечної багатозадачною роботи ПО СТС. Причини багатозадачності. Багатозадачна робота ПЗ і її реалізації при колективній розробці. Завдання, процеси і потоки. Контекст процесу. Узагальнена схема варіантів спільного	2

	використання інформації взаємодіючими процесами. Реалізація багатозадачності за рахунок паралельних обчислень. Технологія Open MP. Закон Амдела Література: [1; 4; 13].	
Тема 5. Захисне конструювання ПЗ, синхронізація процесів, підходи при конструюванні ПЗ (структурний та ООП)		4
9	**Лекція 9. Захисне конструювання ПЗ з паралельними процесами. Задачі синхронізації процесів.** Критичний ресурс ЦОМ. Основне правило захисту критичних ресурсів ЦОМ при конструюванні ПЗ. Завдання синхронізації процесів. Тимчасова і логічна синхронізація. Взаємне виключення процесів. Використання м'ютексів. Завдання синхронізації «Читачі-письменники» Завдання синхронізації. «Обідають філософи» Система IntelThreadChecker (ITC) і типи виявляються нею помилок. Література: [3-4; 6; 14].	2
10	**Лекція 10. Конструювання ПЗ з мінімізацією його складності. Основні поняття структурного та об'єктно-орієнтованого підходу до конструювання ПЗ. Проектування «зверху вниз» і «знизу вгору».** Характеристики ПЗ, які поліпшують процеси його розробки і супроводу. Мінімізація складності ПЗ. Стандартні прийоми в конструюванні. Основні поняття структурного і об'єктно-орієнтованого підходу до конструювання ПЗ. Рефакторинг. Особливості конструювання програм для вбудованих ЦОМ критичних систем. Фіксований розподіл пам'яті. Проектування знизу-вгору і проектування зверху-вниз. Програмні заглушки і їх використання Література: [3; 5; 7].	2
Тема 6. Евристичні принципи конструювання програм та із захистом від помилок		2
11	**Лекція 11. Евристичні принципи конструювання програм. Конструювання ПЗ, пристосованого до змін (готового до проведення змін).** Класи реального світу предметної області і штучні об'єкти. Надмірно великі і неправильно названі класи. Евристичні прийоми конструювання методів, запобігання дублювання коду. Приховування інформації. Дві категорії секретів програм. Надмірне поширення інформації в програмі Захист від переривання програми в «незручний» час Підвищення пристосованості ПЗ до змін (готовності до змін). Література: [1-2; 7; 12].	2
Тема 7. Методи обробки помилок, аварійний захист ПЗ. Технологія налагодження ПЗ. Методи верифікації та валідації програмного забезпечення.		4
12	**Лекція 12. Конструювання програм із захистом від помилок (стійких до помилок програм).** Низькорівневий і високорівневий захист від помилок в програмах. Стійкі до помилок програми Види контролю роботи ПЗ. Контроль роботи ПЗ вбудованими засобами без припинення його функціонування. Еталони для контролю роботи ПЗ. Низькорівневі засоби виявлення помилкового функціонування ПЗ. Виняткові ситуації (Винятки). Література: [5-6, 14, 15].	2
13	**Лекція 13. Методи обробки помилок у вхідних і вихідних даних. Твердження і загальні принципи їх використання Конструювання аварійного захисту в ПЗ для мінімізації збитку від проявилися помилок.** Три основні кроки захисного конструювання. Методи обробки можливих помилок у вхідних і вихідних даних. Що важливіше стійкість або коректність? Твердження і загальні принципи їх використання. Системний рівень захисту від помилок при конструюванні ПЗ. Стратегії безпеки. Три типи реакції ПЗ на виявлену помилку. Відмовостійкі системи. Перелік позаштатних ситуацій. Аварійний захист. Захисне конструювання ПЗ зі скасуванням помилкових команд, виданих з ПЗ або на ПЗ. Література: [3, 4, 16].	2
Тема 8. Відладки при конструюванні ПЗ та контроль за ними. Інструментальні програмні засоби доменного аналізу, проектування, тестування, візуалізації, вимірювань та документування програмного забезпечення.		6

14	**Лекція 14. Технологія налагодження ПЗ. Помилки ПЗ. Статична, динамічна, структурна, функціональна відладки.** Помилки ПЗ, відладка і тестування програмного забезпечення. Два інструментальних засоби відладки. Аналіз виявлених в ПЗ помилок і їх класифікація Статична відладка і динамічна відладка Принцип «білого» і «чорного» ящика при динамічній відладці ПЗ. Функціональна відладка. Література: [4, 7, 12, 18].	2
15	**Лекція 15. Структурна динамічна відладка. Автономна відладка (АВ) і комплексна відладка (КВ) ПЗ. Послідовність дій при відладці ПЗ**. Структурна динамічна відладка. Вибір критеріїв для відбору варіантів роботи ПЗ при відладці Автономна відладка (АВ) і комплексна відладка (КВ) ПЗ. Драйвери і заглушки при автономній відладці. Послідовність дій при відладці ПЗ. Література: [4, 13, 16].	2
16	**Лекція 16. Деякі проектні моделі оцінки числа маршрутів при відладці ПЗ. Контроль відладки ПЗ в процесі налагодження.** Наближений метод оцінки числа варіантів для відладки ПЗ. Графи-дерева, як модель структури ПЗ. Регулярне і випадкове дерево структури ПЗ і стійкість його структурного параметра. Контроль налагодженості ПЗ в процесі відладки. Гіпотеза Джелінского-Моранді і математична модель надійності ПЗ. Приріст кількості помилок, виявлених на інтервалі. Метод найменших квадратів для апроксимації експериментальних даних по помилках ПЗ. Література: [4, 14, 17].	2
Тема 9. Моделювання роботи СТС Успадковане ПЗ. Мобільність ПЗ і реінженіринг ПЗ.		2
17	**Лекція17. Моделювання роботи СТС з метою генерації даних для проведення комплексної відладки ПЗ. Успадковане ПЗ. Мобільність ПЗ і реінжиніринг ПЗ.** Проблема генерації даних на комплексну відладку. Принципи математичного моделювання. Переваги математичного моделювання зовнішнього середовища. Успадковане ПЗ. Мобільність ПЗ і реінжиніринг ПЗ. Емуляція роботи старої ЦОМ на новій апаратній платформі. Література: [5, 7, 16].	2
Разом за 7-й семестр		**34**

5.2 Зміст лабораторних занять

№ з/п	Теми лабораторних занять	Години
1	Планування розробки ПЗ. Етапи розробки програмного забезпечення при структурному підході до програмування. Стадія «Технічне завдання». Передпроєктне обстеження предметної області, системний аналіз об'єкта проектування.	**2**
2	Структурний підхід до програмування. Стадія «Ескізний проект». Сітковий графік виконання робіт. Вибір методології розробки ПЗ. Опис вимог до ПЗ. Робота в команді.	**4**
3	Стадія розробки програмного забезпечення «Технічний проект». Структурний підхід до програмування. Розробка людино-машинного інтерфейсу. Розробка прототипу ПЗ.	**4**
4	Етапи розробки програмного забезпечення Стадія «Реалізація». Проектування та реалізація проекту ПЗ з інтерфейсом користувача.	**4**
5	Виконання оцінки проекту на основі loc- і fp-метрик.	**4**
6	Аналіз чутливості програмного проекту на основі моделі COCOMO II	**6**

7	Метрики об'єктно-орієнтованих програмних систем.	**6**
8	Організація таблиць у трансляторах і робота з ними. Рефакторинг. Оптимізація коду.	**4**
Разом за 7-й семестр		**34**

5.3 Зміст самостійної роботи

Об'єм самостійної роботи з дисципліни **"Конструювання програмного забезпечення"** становить 112 години. Вони включають опрацювання лекційного матеріалу, теоретичних і лабораторних завдань, підготовку до виконання лабораторних робіт, їх захисту і поточне тестування.

Номер тижня	Назва теми	Години
1	**Тема 1. Предмет і зміст дисципліни.** Опрацювання лекційного матеріалу, підготовка до виконання лабораторної роботи №1. Література: [1-5; 7; 11].	6
2	**Тема 1. Предмет і зміст дисципліни.** Опрацювання лекційного матеріалу. Захист лабораторної роботи № 1. Література: [4; 7; 11].	6
3	**Тема 2. Системний підхід до розробки ПЗ. Моделі життєвого циклу ПЗ.** Опрацювання лекційного матеріалу, підготовка до виконання лабораторної роботи №2. Література: [3-4; 10].	6
4	**Тема 2. Системний підхід до розробки ПЗ. Моделі життєвого циклу ПЗ.** Опрацювання лекційного матеріалу. Захист лабораторної роботи №2. Література: [4; 7; 11].	6
5	**Тема 3. Проектування при конструюванні ПЗ.** Опрацювання лекційного матеріалу. Підготовка до виконання лабораторної роботи №3. Література: [4; 7; 11].	6
6	**Тема 3. Проектування при конструюванні ПЗ.** Опрацювання лекційного матеріалу, захист лабораторної роботи №3. Література: [1; 4; 12].	6
7	**Тема 4. Конструювання ПЗ та багатозадачність.** Опрацювання лекційного матеріалу. Підготовка до виконання лабораторної роботи №4. Література: [1; 4; 12].	6
8	**Тема 4. Конструювання ПЗ та багатозадачність.** Опрацювання лекційного матеріалу, захист лабораторної роботи №4. Література: [5-6; 10]. Підготовка до тестування.	6
9	**Тема 5. Захисне конструювання ПЗ, синхронізація процесів, підходи при конструюванні ПЗ (структурний та ООП).** Опрацювання лекційного матеріалу, підготовка до виконання лабораторної роботи №5. Література: [2-4; 8-9]	6
10	**Тема 5. Захисне конструювання ПЗ, синхронізація процесів, підходи при конструюванні ПЗ (структурний та ООП).** Опрацювання лекційного матеріалу, захист лабораторної роботи №5. Література: [4; 6; 13]	6
11	**Тема 6. Евристичні принципи конструювання програм та із захистом від помилок.** Опрацювання лекційного матеріалу, підготовка до виконання лабораторної роботи №6. Література: [1; 3; 6]	6
12	**Тема 6. Евристичні принципи конструювання програм та із захистом від помилок.** Опрацювання лекційного матеріалу, захист лабораторної роботи №6. Література: [2; 13].	6
13	**Тема 7. Методи обробки помилок, аварійний захист ПЗ. Технологія налагодження ПЗ.** Опрацювання лекційного матеріалу, підготовка до виконання лабораторної роботи №7. Література: [4; 7; 13].	6

14	**Тема 7. Методи обробки помилок, аварійний захист ПЗ. Технологія налагодження ПЗ.** Опрацювання лекційного матеріалу. Захист лабораторної роботи №7. Література: [4; 7; 13].	6
15	**Тема 8. Відладки при конструюванні ПЗ та контроль за ними.** Опрацювання лекційного матеріалу,підготовка до виконання лабораторної роботи №8. Література: [2; 5; 11].	6
16	**Тема 8. Відладки при конструюванні ПЗ та контроль за ними.** Опрацювання лекційного матеріалу, виконання лабораторної роботи №8. Література: [2; 5; 11].	6
17	**Тема 9. Моделювання роботи СТС Успадковане ПЗ. Мобільність ПЗ і реінженіринг ПЗ.** Опрацювання лекційного матеріалу, захист лабораторної роботи №8. Підготовка до іспиту	16
Разом за 7-й семестр		**112**

6. МЕТОДИ НАВЧАННЯ

Процес навчання з дисципліни ґрунтується на використанні традиційних та сучасних методів: методи проблемного викладання, словесні, наочні (лекції); пояснювально-ілюстративні, проблемного викладання, дослідницькі, частково-пошукові (лабораторні заняття), проблемного викладання, дослідницькі, частково-пошукові (самостійна робота: індивідуальні завдання). Всі заняття проводяться з використанням інформаційних технологій і мають за мету – набуття здобувачами першого (бакалаврського) рівня вищої освіти практичних навичок з конструювання програмного забезпечення, тестування та налагодження відладки, використовувати метрики для оцінки розробленого ПЗ.

7. ФОРМИ І МЕТОДИ ОЦІНЮВАННЯ РЕЗУЛЬТАТІВ НАВЧАННЯ

Поточний контроль здійснюється під час лекційних та лабораторних занять. Семестровий контроль проводиться у формі іспиту. При виведенні остаточної оцінки враховуються результати поточного контролю та письмового іспиту.

Процес оцінювання підготовленості здобувача першого (бакалаврського) рівня вищої освіти можна розділити на етапи:

Перший етап оцінювання направлений на визначення знань інформаційного мінімуму. Якщо здобувач твердо засвоїв визначену навчальним планом суму формальних знань, то це означає, що він вміє використати їх при вирішенні різних питань при проектуванні програмних систем, вміє розширити їх.

Перед вивченням дисципліни, як правило, проводиться вхідний контроль знань з дисциплін, що їй передують і забезпечують. При цьому необхідно встановити рівні та критерії сформованості знань щодо змісту навчальних елементів. Такими рівнями є:

Ознайомчо-орієнтовний (ОО) – особа має орієнтовне уявлення щодо понять, які вивчаються, здатна: програмувати основні елементи програмних систем різними мовами програмування, обирати сучасні методології та технології проектування програмного забезпечення, обґрунтовано використовувати сучасні середовища розроблення програмного забезпечення для розроблення програмних систем.

Понятійно-аналітичний (ПА) – особа має чітке уявлення щодо навчального об'єкту, здатна перенести раніше засвоєнні знання на типові ситуації.

Продуктивно-синтетичний (ПС) – особа має глибоке розуміння щодо навчального об'єкту, здатна здійснювати синтез, генерувати нові ідеї та уявлення, переносити раніше засвоєні знання на нетипові, нестандартні ситуації.

Кожний вид роботи з дисципліни оцінюється за *чотирибальною* шкалою. Семестрова підсумкова оцінка визначається як середньозважена з усіх видів навчальної роботи, виконаних і зданих *позитивно* з врахуванням коефіцієнта вагомості. Вагові коефіцієнти змінюються залежно від структури дисципліни і важливості окремих її видів робіт. Здобувач, який набрав позитивний середньозважений бал за поточну роботу і не здав контрольний захід (іспит), вважається невстигаючим.

При оцінюванні знань здобувачів першого (бакалаврського) рівня вищої освіти використовуються різні засоби контролю, зокрема: усне опитування перед допуском до виконання лабораторної роботи – здійснюється на її початку; засвоєння теоретичного матеріалу з тем перевіряється тестовим контролем;

якість виконання, набуття теоретичних знань і практичних навичок перевіряється шляхом захисту кожної лабораторної роботи згідно з робочою програмою дисципліни і робочим навчальним планом.

Оцінка, яка виставляється за *лабораторне заняття*, складається з таких елементів: усне опитування здобувачів перед допуском до виконання лабораторної роботи; знання теоретичного матеріалу з теми; якість оформлення протоколу і графічної частини; вміння здобувача обґрунтувати прийняті конструктивні рішення; своєчасний захист лабораторної роботи. Для виконання програми дисципліни здобувач повинен отримати 7 оцінок за лабораторні роботи.

Термін захисту лабораторної роботи вважається своєчасним, якщо здобувач захистив її на наступному після виконання роботи занятті.

Пропущене лабораторне заняття здобувач повинен відпрацювати не пізніше, ніж за два тижні до кінця теоретичних занять у семестрі.

При *оцінюванні знань* здобувачів першого (бакалаврського) рівня вищої освіти викладач керується такими критеріями.

Оцінку «відмінно» отримує здобувач за глибоке і повне опанування змісту навчального матеріалу, в якому він легко орієнтується, понятійного апарату, за уміння зв'язувати теорію з практикою, вирішувати практичні завдання, висловлювати і обґрунтовувати свої судження. Відмінна оцінка передбачає грамотний, логічний виклад відповіді (як в усній, так і в письмовій формі), якісне зовнішнє оформлення. Здобувач повинен набути практичних навичок із проектування та програмної реалізації програмних систем.

Оцінка «відмінно» виставляється здобувачу, який глибоко засвоїв основні принципи проектування програмних систем та вміє їх раціонально застосувати, знає методики та вміє ними користуватися при розробленні програмного забезпечення. Здобувач не повинен вагатися при видозміні запитання, повинен робити детальні та узагальнюючі висновки.

Оцінку «добре» отримує здобувач за повне засвоєння навчального матеріалу, володіння понятійним апаратом, орієнтування у вивченому матеріалі, свідоме використання знань для вирішення практичних завдань, грамотний виклад відповіді, але у змісті і формі відповіді мали місце окремі неточності (похибки), нечіткі формулювання закономірностей тощо. Відповідь здобувача повинна будуватись на основі самостійного мислення.

Оцінку «добре» отримує здобувач за правильну відповідь з однією-двома суттєвими помилками.

Оцінки «задовільно» заслуговує здобувач, який виявив знання основного навчально-програмного матеріалу в обсязі, необхідному для подальшого навчання та практичної діяльності за професією, що справляється з виконанням практичних завдань, передбачених програмою. Як правило, відповідь здобувача будується на рівні репродуктивного мислення, здобувач слабо знає структуру курсу, допускає помилки у відповіді, засвоїв і набув практичних навичок у проектуванні та реалізації програмних систем, але припустився неточностей. Вагається при відповіді на видозмінене запитання, разом з тим здобувач володіє знаннями, що дозволяють йому під керівництвом викладача усунути неточності у відповіді.

Оцінки «задовільно» заслуговує здобувач за неповне опанування програмного матеріалу, але отримані знання і набуті практичні навички із проектування та розроблення програмного забезпечення.

Оцінка «незадовільно» виставляється, коли здобувач має розрізнені, безсистемні знання, не вміє виділяти головне і другорядне, допускається помилок у визначенні понять, перекручує їх зміст, хаотично і невпевнено викладає матеріал, не може використовувати знання при вирішенні практичних завдань. Як правило, оцінка "незадовільно" виставляється здобувачу, який не може продовжити навчання без додаткових знань з курсу.

На основі результатів поточного контролю і іспиту виставляється підсумкова семестрова оцінка.

Залік виставляється при отриманні здобувачем з дисципліни від 3,00 до 5,00 балів. При цьому за вітчизняною шкалою ставиться «зараховано», а за шкалою ECTS – набрана здобувачем кількість балів та відповідна їй оцінка.

При викладанні дисципліни використовуються такі види навчальних занять, як лекції, лабораторні роботи, індивідуальне консультування і керівництво самостійною роботою здобувача, в т.ч. за індивідуальним завданням.

При оцінюванні знань здобувачів використовуються різні засоби контролю, зокрема: допуск до виконання лабораторної роботи здійснюється на її початку усним опитуванням кожного здобувача; засвоєння теоретичного матеріалу змістових модулів перевіряється змістовим контролем; якість виконання, набуття теоретичних знань і практичних навичок перевіряється шляхом захисту кожної лабораторної роботи та індивідуального завдання згідно з робочим планом.

Оцінка, яка виставляється за лабораторне заняття, складається з таких елементів: усне опитування здобувачів перед допуском до виконання лабораторної роботи; знання теоретичного матеріалу з теми; якість оформлення протоколу і графічної частини; вміння здобувача обґрунтувати прийняті конструктивні рішення; своєчасний захист лабораторної роботи.

Термін захисту лабораторної роботи вважається своєчасним, якщо здобувач захистив її на наступному після виконання роботи занятті.

Пропущене з поважної причини лабораторне заняття здобувач повинен відпрацювати в лабораторіях кафедри у встановлений викладачем термін.

Система поточного контролю знань, умінь, навичок

Формою поточного контролю, що проводиться на кожному лабораторному занятті, є коротке усне та письмове опитування викладеного лекційного матеріалу, а також письмове виконання прикладів розв'язування задач. Формою підсумкового контролю є письмова контрольна робота, яка включає одне питання з переліку питань для підсумкового контролю і задачу.

Структурування дисципліни за видами робіт і оцінювання результатів навчання здобувачів у семестрі за ваговими коефіцієнтами

<table>
<tr><td colspan="9">Аудиторна робота</td><td rowspan="2">Семестровий контроль, іспит (І)</td><td rowspan="2">Підсумковий бал</td></tr>
<tr><td colspan="8">Лабораторні роботи (ЛР)</td><td>Тест</td></tr>
<tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>Т</td><td rowspan="2">0,4</td><td rowspan="2">ЛР*0,3+Т*0,3+І*0,4</td></tr>
<tr><td colspan="8">ВК = 0,3</td><td>ВК = 0,3</td></tr>
</table>

Умовні позначення: ВК – ваговий коефіцієнт, ЛР – лабораторна робота, Т – тест, І – іспит.

Оцінювання тестових завдань. Тематичний тест для кожного здобувача складається з двадцяти тестових завдань, кожне з яких оцінюється одним балом. Максимальна сума балів, яку може набрати здобувач, складає 20.

Оцінювання здійснюється за чотирибальною шкалою.

Відповідність набраних балів за тестове завдання оцінці, що виставляється здобувачу, представлена у нижченаведеній таблиці.

Сума балів за тестове завдання	1–11	12–14	15–18	19-20
Оцінка	2	3	4	5

На тестування відводиться 20 хвилин. Тестування проводиться з використанням модульного середовища для навчання MOODLE. Правильні відповіді здобувач реєструє в он-лайн режимі в модульному середовищі MOODLE. Через 30 хвилин здобувачі завершують тестування та надсилають свої відповіді на сервер. Викладач оголошує результати тестування згідно журналу оцінок модульного середовища MOODLE.

Якщо здобувач отримав негативну оцінку, то він має перездати її в установленому порядку, але обов'язково до терміну наступного контролю.

Підсумкова семестрова оцінка за національною шкалою і шкалою ЄКТС встановлюється в автоматизованому режимі після внесення усіх оцінок до електронного журналу. Співвідношення вітчизняної шкали оцінювання і шкали оцінювання ЄКТС наведені у наступній таблиці.

Співвідношення вітчизняної шкали оцінювання і шкали оцінювання ЄКТС

Оцінка ЄКТС	Інституційна інтервальна шкала балів	Вітчизняна оцінка, критерії		
A	4,75–5,00	5	***Відмінно*** – глибоке і повне опанування навчального матеріалу і виявлення відповідних умінь та навиків	Зараховано
B	4,25–4,74	4	***Добре*** – повне знання навчального матеріалу з кількома незначними помилками	

C	3,75–4,24	4	***Добре*** – в загальному правильна відповідь з двома-трьома суттєвими помилками	
D	3,25–3,74	3	***Задовільно*** – неповне опанування програмного матеріалу, але достатнє для практичної діяльності за професією	
E	3,00–3,24	3	***Задовільно*** – неповне опанування програмного матеріалу, що задовольняє мінімальні критерії оцінювання	
FX	2,00–2,99	2	***Незадовільно*** – безсистемність одержаних знань і неможливість продовжити навчання без додаткових знань з дисципліни	Незараховано
F	0,00–1,99	2	***Незадовільно*** – необхідна серйозна подальша робота і повторне вивчення дисципліни	

Залік виставляється, якщо середньозважений бал, який отримав здобувач з дисципліни, знаходиться у межах від 3,00 до 5,00 балів. При цьому за вітчизняною шкалою ставиться оцінка «зараховано», а за шкалою ЄКТС – буквене позначення оцінки, що відповідає набраній здобувачем кількості балів відповідно до таблиці Співвідношення.

8. ПИТАННЯ ДЛЯ САМОКОНТРОЛЮ

1. Каскадні моделі життєвого циклу розробки програмних систем
2. Діаграми пакетів в UML
3. Спіралевидні моделі життєвого циклу розробки програмних систем
4. Способи задання обмежень в UML діаграмах.
5. Спіралевидні моделі життєвого циклу розробки об’єктно-орієнтованих програмних систем
6. Типи відношень, які використовуються в класових діаграмах
7. Роботи, що проводяться на етапі аналізу для життєвого циклу розробки програмних систем.
8. Способи відображення активних об’єктів в UML діаграмах.
9. Роботи, що виконуються при специфікації вимог до програмних систем.
10. Відношення спадкування в діаграмах класів.
11. Роботи, що виконуються на етапі системного аналізу
12. Відношення агрегації в класових діаграмах.
13. Роботи, що виконуються на етапі об’єктного аналізу.
14. Відношення асоціації в діаграмах класів та способи їх реалізації в об’єктно-орієнтованих мовах програмування
15. Роботи, що виконуються на етапі проектування.
16. Способи представлення багатошарових архітектур засобами UML.
17. Роботи, що виконуються на етапі проектування механізмів.
18. Подібності і відмінності автоматних моделей і діаграм Харела, що використовуються в діаграмах станів в UML.
19. Роботи, що виконуються на етапі детального проектування.
20. Параметризація класів в UML.
21. Роботи, що виконуються на етапі проектування архітектури.
22. Типи обмежень у вбудованих системах та системах реального часу.
23. Роботи, що виконуються на етапі кодування.
24. Використання UML-діаграм на різних етапах життєвого циклу розробки програмної системи.
25. Роботи, що виконуються на етапі тестування.
26. Мови специфікацій вимог до програмного забезпечення.
27. Діаграми прецедентів (UseCase діаграми) в UML.
28. Поняття мета класу.
29. Об’єктні діаграми (діаграми взаємодії) в UML.
30. Типи відношень в UseCase діаграмах.
31. Діаграми класів в UML.
32. Інструментальні засоби проектування, які використовують UML в якості вхідної мови.
33. Діаграми станів в UML.

34. Сценарії для UseCase діаграм та способи їх представлення.
35. Діаграми послідовностей в UML.
36. Відношення асоціації та його використання при генерації програм.
37. Діаграми розгортання в UML.
38. Способи задавання обмежень в діаграмах послідовностей в UML.
39. Діаграми компонентів в UML.
40. Класові діаграми та їх роль в діаграмах розгортання.
41. Проведення оцінки проекту на основі loc- і fp-метрик.
42. Аналіз чутливості програмного проекту на основі моделі COCOMO II
43. Принцип дії рефакторингу.
44. Метрики об'єктно-орієнтованих програмних систем.
45. Оптимізація коду, принцип дії.

9. МЕТОДИЧНЕ ЗАБЕЗПЕЧЕННЯ

Навчальний процес з дисципліни " Конструювання програмного забезпечення " повністю і в достатній кількості забезпечений необхідною навчально-методичною літературою (усі необхідні навчально-методичні матеріали розміщені у модульному навчальному середовищі).

10. РЕКОМЕНДОВАНА ЛІТЕРАТУРА

ОСНОВНА:

1. Роберт Мартін. Чистий код: Чистий код: створення, аналіз, рефакторинг. - Фабула. 2019 – 416 с.
2. Постіл. С.Д. UML. Уніфікована мова моделювання інформаційних систем: Навч. посіб., 2019. - 321 с.
3. Piotr Sliż. Organizacja procesowo-projektowa. Istota, modelowanie, pomiar dojrzałości. Difin – 2021 – 292 str.
4. Ian Sommerville. Software Engineering, 10th edition. Published by Pearson . July 14th 2021 – 816 p.
5. Philippe Kruchten. The Rational Unified Process: An Introduction (5rd Edition) (Addison-wesley Object Technology Series). Addison-Wesley Professional; 5rd edition. 2018 – 407 p.
6. Pinto J. Project Management: Achieving Competitive Advantage, 5th Edition. – 2019. – 592 p.
7. Martin Fowler. Refactoring: ImprovingtheDesignofExistingCode (WebEdition), 2nd Edition/ WebEdition.- 2018

ДОПОМІЖНА:

8. Shyam R Chidamber,Chris F Kemerer (Author), Sloan School of Management. A Metrics Suite for Object Oriented Design. Franklin Classics. - 2018 – 44 p.
9. D.Spinellis.Programcodeanalysis -AddisonWesley, 2004
10. ErichGamma, RichardHelm, RalphJohnson, andJohnVlissides. DesignPatterns: ElementsofReusable Object-Oriented Software.Addison-Wesley - 2009 - 417p.
11. Beck Kent. Extreme Programming Explained: EmbraceChange (The XP Series). Addison-WesleyProfessional. – 2004 - 224
12. Dave Nicolette. Software Development Metrics 1st Edition - Manning; 1st edition – 2015 – 192 p.
13. Eric J. Braude; Michael E. Bernstein. Software Engineering. Waveland Press, 2016 – 802 p.
14. Bohm, Corrado; andGiuseppeJacopini (May 1966). "FlowDiagrams, TuringMachinesandLanguageswithOnlyTwoFormationRules". Communicationsofthe ACM 9 (5): 366–371. doi:10.1145/355592.365646
15. Dijkstra, E. W. (Aug 1972). "The Humble Programmer". Communications of the ACM 15 (10):

859–866. doi:10.1145/355604.361591.
16. http://www.cs.utexas.edu/~EWD/transcriptions/EWD03xx/EWD340.html. (EWD340) PDF, 1972 ACM TuringAwardlecture
17. RussellGold, ThomasHammell, TomSnyder. TestDrivenDevelopment: A J2EE Example.- Apress, 2005.- 296 pages.
18. Software design, architecture and engineering: concepts and practice. (2021). (n.p.): PHI Learning Pvt. Ltd.. - 280p.
19. Voorhees, D. P. (2021). Guide to Efficient Software Design: An MVC Approach to Concepts, Structures, and Models. Springer International Publishing. - 519p.
20. Sufyan bin Uzayr, Software Design Patterns: The Ultimate Guide (2021) CRC Press, - 454 p
21. Hanson, C., Sussman, G. J. (2021). Software Design for Flexibility: How to Avoid Programming Yourself Into a Corner. MIT Press. - 519p.
22. Parikh, K., Johri, A. (2022). Combining DataOps, MLOps and DevOps: Outperform Analytics and Software Development with Expert Practices on Process Optimization and Automation (English Edition). BPB Publications. – 400p
23. Bhatt, P. C. P. (2020). Software Design: Concepts and Practice. (n.p.): Independently Published. – 318p
24. Beningo, J. (2022). Embedded Software Design: A Practical Approach to Architecture, Processes, and Coding Techniques. : Apress. . – 463p
25. Chatterjee, S., Deshpande, S., Been, H., Gaag, M. v. d. (2022). Designing and Implementing Microsoft DevOps Solutions AZ-400 Exam Guide: Prepare for the Certification Exam and Successfully Apply Azure DevOps Strategies with Practical Labs. Packt Publishing. – 490p
26. Chin, S., McKay, M., Ruiz, I., Sadogursky, B. (2022). DevOps Tools for Java Developers: Best Practices from Source Code to Production Containers. O'Reilly Media, Incorporated. – 322 p
27. Праворська Н.І., Яшина О.М., Нетреба І.В. Доміна А.Р. Кириченко О.М. Метод конструювання програмного забезпечення згідно аналізу помилок SQL-запитів. Вісник Хмельницького національного університету, серія Технічні науки. №3, 2023 (321) – с. 302-307
28. Праворська Н.І., Мартинюк В.В. Конструювання програмного забезпечення за допомогою синхронного підходу: основні процеси та інструменти для ефективної реалізації devops. Вісник Хмельницького національного університету, Том 1, №5, 2023 (325) – стор.182-192

11. ІНФОРМАЦІЙНІ РЕСУРСИ

1. Модульне середовище для навчання. Доступ до ресурсу: https://msn.khmnu.edu.ua.
2. Електронна бібліотека університету. Доступ до ресурсу: http://library.khmnu.edu.ua
3. Репозитарій ХНУ:https://elar.khmnu.edu.ua/home

KHMELNYTSKYI NATIONAL UNIVERSITY

APPROVED
Dean of the Faculty
Savenko O.S.
" 1 " 09 2023.

COURSE PROGRAM
Software Construction

Field of study: 12 - Information Technologies
Major: 121 – Software Engineering
Level of Higher Education: First Level (Bachelor)
Educational program: Software Engineering
Discipline status: Compulsory
Faculty: Information Technologies
Department: Higher Mathematics and Computer Applications

Study mode	Year	Semester	Total Credits	Number of hours						Course project	Coursework	Semester control form	
				Classwork hours				Seminar classes	Independent work, including individual			pass/ fail test	Exam
			ECTS credits	Total	Lectures	Laboratory works	Practical classes						
Full-time (Daytime)	4	7	6	180	34	34			112				+
Total			**6**	**180**	**34**	**34**			**112**				**1**

The course program is based on the Higher Education Standard, the 2023 Bachelor's degree educational program, and the curriculum.

Program's author N. Pravorska

Approved at the staff meeting of the Software Engineering Department

Minutes from 31.08.2023 No. 1

Head of the Software Engineering Department L. Bedratyuk

The course program is approved by the Academic Board of the Faculty of Information technologies

Head of the Academic Board O.S. Savenko

Khmelnytskyi 2023

SOFTWARE DESIGN

Type of Discipline	Compulsory
Level of Higher Education	First (Bachelor's)
Language of Instruction	Ukrainian, English
Semester	7
ECTS Credits	6
Course study mode	Full-time (Daytime)

Learning outcomes. According to the Standard of higher education and the educational program, the discipline must ensure: **competences *:*** the ability to apply knowledge in practical situations; ability to work in a team; the ability to participate in software design , including modeling (formal description) of its structure, behavior and functioning processes; to accumulate, process and systematize professional knowledge about creating and maintaining software and recognizing the importance of lifelong learning; implement phases and iterations of the life cycle of software systems and information technologies based on relevant models and software development approaches; the ability to carry out the system integration process, apply change management standards and procedures to maintain the integrity, overall functionality and reliability of the software ; Ability to work in parallel on the stages of the software life cycle and effective interaction between the executors of the stages. **program learning outcomes:** to analyze, purposefully search for and choose information and reference resources and knowledge necessary for solving professional tasks, taking into account modern achievements of science and technology; to be able to choose and use the software creation methodology appropriate to the task; conduct a pre-project survey of the subject area, system analysis of the design object; select initial data for design, guided by formal requirements description and modeling methods; apply effective software design approaches in practice; know and apply methods of developing algorithms, designing software and data and knowledge structures; apply in practice instrumental software tools for domain analysis, design, testing, visualization, measurement and software documentation; have skills in team development, approval, design and release of all types of software documentation; be able to apply methods of component software development; know and be able to apply software verification and validation methods . To be able to effectively cooperate with the executors of different stages of the software life cycle, coordinating simultaneous work on the stages. To know and be able to apply techniques, tools and strategies for managing resources, time and communication to ensure effective parallel work on different stages of the software life cycle **Course content.** Elements of software design . Key design principles . Construction tools . Description languages, representations, architectural frameworks. Architecture modeling technique. Identification of goals and key scenarios. A systematic approach to software development. Software life cycle models. Software creation methodology . Identification of key issues. Software design and multitasking. Protective software design, process synchronization, software design approaches. Debugging during software design and their control.

Planned educational activities: 34 hours of lectures, 34 hours of laboratory classes, 112 hours of independent work; together 180 hours

Teaching methods: methods of problem-based teaching, verbal, visual (lectures); explanatory and illustrative, problem-based teaching, research, partially research-based (laboratory classes), problem-based teaching, research, partially research-based.

Assessment forms and methods: oral survey, defense of laboratory works. written independent and control works, written exam

Semester control form : exam

Educational resources:

1. Robert Martin. Clean Code: Clean Code: Build, Analyze, Refactor . - A story. 2019 – 416 p.
2. The bed S.D. UML. Unified modeling language: Teaching . manual , 2019. - 321 p.
3. Aho , Alfred V. fnd Jeffrey D. Ullman . The Theory of Parsing , Translation , and Compiling (Volume 2: Compiling). -Prentice-Hall. -2018
4. Ian Sommerville . Software Engineering , 10th edition . Published by Pearson . July 14th 2021 - 816

Lecturer: PhD, Associate Professor Pravorska N.I.

3. EXPLANATORY NOTE

The discipline " Software Design " is a discipline of professional and practical training.

The goal of the discipline is to form students' theoretical knowledge and practical skills in software design, testing and debugging , to use metrics to evaluate the developed software. The discipline is aimed at studying the basic principles and methodologies of software design, including software life cycle models, design technologies, as well as means and methods of creating and implementing projects.

The subject of the discipline: models of the software life cycle and the main methodologies and means of software development. Technology of creating and documenting software products.

Objects the study includes: design technologies, models and methods of software life cycle support, means and methods of creating and implementing projects.

The main tasks are: providing the bachelor's degree holder with the opportunity to gain knowledge about functional and object-oriented design technologies; formation of practical skills for creating software using databases; training of applicants of the first (bachelor's) level of higher education to study other information technology disciplines.

Tasks of the discipline: familiarization of applicants of the first (bachelor's) level of higher education with modern methods and technologies of constructive software; studying methods of designing software using the UML modeling language; forming skills and abilities to produce design solutions; formation of work skills in modern instrumental environments supporting the process of designing software systems.

According to the Higher Education Standard for the specified specialty and educational program, the discipline "Software Design" must ensure the formation of professional and general competencies and program results, namely:

Integral competence

The ability to solve complex specialized tasks or practical problems of software engineering, characterized by complexity and uncertainty of conditions, with the application of theories and methods of information technology.

General competences :

GC1. Ability for abstract thinking, analysis, and synthesis.

GC6. Ability to search, process, and analyse information from various sources.

Professional competences.

PC2. Ability to participate in software design, including modelling (formal description) of its structure, behaviour, and operational processes.

PC10. Ability to accumulate, process, and systematise professional knowledge regarding the creation and maintenance of software and recognise the importance of lifelong learning.

PC11. Ability to implement phases and iterations of the life cycle of software systems and information technologies based on relevant software development models and approaches.

PC12. Ability to execute the system integration process and apply standards and change management procedures to maintain the integrity, overall functionality, and reliability of the software.

PC15. Ability for parallel work on stages of the software lifecycle and effective interaction between stage performers.

***program learning outcomes* :**

PLO1 To analyse, purposefully search for, and select the necessary information, reference resources, and knowledge for solving professional. PLO6 To select and utilise a software development methodology appropriate for the task. PLO8 To have the skills to develop a human-machine interface. PLO9 To know and be able to use methods and tools for collecting, formulating, and analysing software requirements. PLO10 To conduct a pre-project survey of the subject area and system analysis of the design object. PLO11 To select initial data for design, guided by formal methods of requirement descriptions and modelling. PLO12 To apply effective software design

approaches in practice. PLO13 To know and apply methods for algorithm development, software design, and data and knowledge structures. PLO14 To use instrumental software tools in practice for domain analysis, design, testing, visualisation, measurement, and software documentation. PLO16To possess skills in team development, approval, design, and release of all types of software documentation. PLO17 To be skilled in applying methods of component software development. PLO19To know and apply methods for software verification and validation. PLO25 To have the ability to effectively collaborate with performers from different stages of the software lifecycle, coordinating simultaneous work on stages. PLO26 To know and apply methodologies, tools, and strategies for resource, time, and communication management to ensure effective parallel work on various software lifecycle stages.

Learning outcomes. A student who has successfully completed the study of the discipline must: master *professional* terminology and basic concepts of software and its design, develop, *analyze and design* software; *to have* the skills of designing software and evaluating the quality of designing software systems; *know* software design methods; *to be able* to solve tasks that arise at different phases of the life cycle of software systems related to software design; *develop* high-quality and flexible software systems; *conduct* refactoring of software systems

Discipline Policy. The organization of the educational process for the discipline complies with the requirements of the provisions on organizational and instructional-methodological support of the educational process, the educational program, and the curriculum. Students are required to attend lectures, practical classes, laboratory work, etc., according to the schedule, not to be late for classes, and to complete all tasks and checkpoints according to the schedule. Missed practical classes and laboratory work must be independently completed by the student in full and reported to the instructor no later than one week before the next assessment. For practical classes and laboratory work, students must prepare on the relevant topic and demonstrate active participation. Knowledge acquired by an individual in the discipline or its specific sections through informal education is credited according to the Regulation on the procedure for transferring learning outcomes and determining academic differences at KhNU.

4. THE STRUCTURE OF CREDIT CREDITS OF THE DISCIPLINE

Topics	Number of hours for:		
	Lectures	**Lab. works**	**Individual work**
Topic 1. Subject and content of the discipline. The place of discipline in the structure of education. Software development problems and ways to solve them. Software engineering	2	2	12
Topic 2. Systematic approach to software development. Software life cycle models. Software creation methodology	4	4	12
Topic 3. Designing when designing software.	8	4	12
Topic 4. Software design and multitasking.	2	4	12
Topic 5. Protective software design, process synchronization, software design approaches (structural and OOP)	4	4	12
Topic 6. Heuristic principles of program design and error protection	2	4	12
Topic 7. Error handling methods, software emergency protection. Software debugging technology. Software verification and validation methods.	4	4	12
Topic 8. Debugging during software design and their control. Instrumental software tools for domain analysis, design, testing, visualization, measurement and software documentation.	6	4	12
Topic 9. Modeling the work of CTS Legacy software. Software mobility and software reengineering.	2	4	16
Total	**34**	**34**	**112**

5. COURSE PROGRAM

5.1. Content of lectures

No lectures	List of topics of lectures, their annotations	Number of hours
Topic 1. Subject and content of the discipline. The place of discipline in the structure of education. Software development problems and ways to solve them. Software engineering		2
1	**Lecture 1. Software development problems and ways to solve them. Software engineering. Solving professional problems with the help of modern achievements of science and technology.** Programming given examples and solving problems using software - what is the difference? The role of software and computers in production, social life and science. Software engineering. Software development problems and ways to solve them. Analysis of the subject area and search for solutions to design problems with the help of modern achievements of science and technology. Pre-project examination of the subject area, system analysis of the design object, selection of information and reference resources necessary for solving professional tasks, taking into account modern achievements of science and technology. Literature: [1-3; 7; 12].	2
Topic 2. Systematic approach to software development. Software life cycle models. Software creation methodology.		4
2	**Lecture 2. Software quality and factors affecting it. Methods of component software development.** Software development technology and software quality.	2

	The choice of software creation methodology. Software quality characteristics that are important to the user. Factors affecting software quality . Literature: [2; 5; 13].	
3	**Lecture 3. A systematic approach to software development. Cascade model of the software life cycle. Software design. Developer software quality criteria. Standards and software development.** Temporal and "spatial" aspects of the system approach Stages of the software life cycle. Cascade model of the life cycle. Software design and internal software quality criteria are developer criteria. Software development standards. Types and meaning of standards, requirements of standards. Selection of input data for design based on formal requirements description and modeling methods. Three groups of software creation processes. Literature: [3-4; 11, 14].	2
Topic 3. Designing when designing software.		8
4	**Lecture 4. Software verification. V-shaped scheme of the software life cycle. spiral model of LC Software. Software development technologies: "heavy and light", Agile methodology , HR programming, DevOps (synchronous execution of stages of development).** Software life cycle and verification processes. Testing, verification, validation V-shaped model of the software life cycle Spiral model of software life cycle. "Hard and fast" software development technologies. Agile methodology (teamwork). Extreme (XR) programming. DevOps technologies (tools and strategies for managing resources, time and communication - synchronous execution of the stages of development). Literature: [4; 7; 8-10, 15].	2
5	**Lecture 5. Parallel work on the stages of the software life cycle.** The concept and importance of parallel work in the software life cycle. Advantages and challenges of parallel execution of work at different stages of development. Methods and tools for coordinating parallel work, including Agile and Scrum approaches. Techniques for effective interaction between teams at various stages of the software life cycle, including planning, development, testing, and implementation. The role of DevOps in supporting parallel work and enabling continuous integration/continuous deployment (CI/CD). Practical examples of using parallel work to reduce development time and improve software quality. Literature: [2; 4-6; 1 6].	2
6	**Lecture 6. Communication strategies and tools for effective interaction among the participants of software development stages.** Overview of the main communication models and their application in the context of software development. The importance of feedback and its role in parallel work. Collaboration and communication technologies, such as project management systems, chats, video conferences, and other teamwork tools. Challenges and best practices in conflict management and misunderstandings between teams. Methods for building effective inter-team relations and a culture of cooperation in multidisciplinary projects. Examples of successful interaction between teams at different stages of the software life cycle. Literature: [1-3; 7; 16]	2
7	**Lecture 7. Software structure diagram. CTS as a means of designing software. Hierarchical structure of CTS software. Cyclicity (periodicity) in the time of solving software tasks. Cooperation between the performers of various stages of the LC software.** System structure, management hierarchy and software structure. Development of the system and implementation of changes in the hierarchical structure of software. Cyclicality (periodicity) in the time of solving management tasks and software operation. Cooperation of development teams and operations teams at various stages of the LC during software development. Literature: [4-5; 6]	2
Topic 4. Software design and multitasking.		2
8	**Lecture 8. A temporary diagram of system operation as a means of designing software that provides parallel physical processes. Software multitasking**	2

	and its causes. Temporary diagram of system and software operation with parallel physical processes. The problem of sharing resources for safe multitasking of CTS software. Reasons for multitasking. Multitasking software and its implementation during collective development. Tasks, processes and flows. Process context. Generalized scheme of options for sharing information by interacting processes. Implementation of multitasking due to parallel computing. Open MP technology. Amdel's law Literature: [1; 4; 13].	
Topic 5. Protective software design, process synchronization, software design approaches (structural and OOP)		4
9	**Lecture 9. Protective design of software with parallel processes. Tasks of process synchronization.** A critical resource of the Center. The main rule of protection of critical resources of COM when designing software. Process synchronization task. Temporal and logical synchronization. Mutual exclusion of processes. Use of mutexes. Synchronization task "Readers-writers" Synchronization task. "Philosophers dine" The IntelThreadChecker (ITC) system and the types of errors detected by it. Literature: [3-4; 6; 14].	2
10	**Lecture 10. Software design with minimization of its complexity. Basic concepts of structural and object-oriented approach to software design. Top-down and bottom-up design.** Features of the software that improve the processes of its development and maintenance. Minimization of software complexity. Standard techniques in construction. Basic concepts of structural and object-oriented approach to software design. Refactoring. Peculiarities of designing programs for built-in central control systems of critical systems. Fixed memory allocation. Bottom-up design and top-down design. Program stubs and their use Literature: [3; 5; 7].	2
Topic 6. Heuristic principles of program design and error protection		2
11	**Lecture 11. Heuristic principles of program design. Designing software adapted to changes (ready for changes).** Classes of the real world of the subject area and artificial objects. Overly large and incorrectly named classes. Heuristic methods of method construction, code duplication prevention. Hiding information. Two categories of program secrets. Excessive dissemination of information in the program Protection against interrupting the program at an "inconvenient" time Increasing the software's adaptability to changes (readiness for changes). Literature: [1-2; 7; 12].	2
Topic 7. Error handling methods, software emergency protection. Software debugging technology. Software verification and validation methods .		4
12	**Lecture 12. Designing programs with protection against errors (error-tolerant programs).** Low-level and high-level protection against program errors. Error-resistant programs Types of software control. Control of software operation by built-in means without stopping its operation. Benchmarks for controlling software operation. Low-level means of detecting software malfunctions. Exceptional situations (Exceptions). Literature: [5-6, 14 , 15].	2
13	**Lecture 13. Methods of handling errors in input and output data. Assertions and general principles of their use Construction of emergency protection in software to minimize damage from errors.** Three main steps of protective construction. Methods of processing possible errors in input and output data. What is more important stability or correctness? Claims and general principles of their use. The system level of protection against errors in software design. Security strategies. Three types of software response to a detected error. Fault-tolerant systems. List of freelance situations. Emergency protection. Defensive design of the software with cancellation of erroneous commands issued from the software or to the software. Literature: [3 , 4 , 16].	2
Topic 8. Debugging during software design and their control. Instrumental software for domain analysis, design, testing, visualization, measurement, and software		6

documentation.		
14	**Lecture 14. Software debugging technology. Software errors. Static, dynamic, structural, functional debugging.** Software errors, software debugging and testing. Two debugging tools. Analysis of errors found in the software and their classification Static debugging and dynamic debugging The principle of "white" and "black" boxes in dynamic software debugging. Functional debugging. Literature: [4, 7, 12, 18].	2
15	**Lecture 15. Structural dynamic debugging. Autonomous debugging (AB) and complex debugging (KB) of software. Sequence of actions during software debugging** . Structural dynamic debugging. Selection of criteria for selecting options for software operation during debugging Autonomous debugging (AB) and complex debugging (CB) of software. Drivers and plugs for offline debugging. Sequence of actions during software debugging. Literature: [4 , 13, 16].	2
16	**Lecture 16. Some design models for estimating the number of routes during software debugging. Software debugging control during debugging.** An approximate method of estimating the number of options for software debugging. Tree graphs as a model of software structure. Regular and random tree of software structure and stability of its structural parameter. Control of the correctness of the software during the debugging process. Dzhelinsky-Morandi hypothesis and mathematical model of software reliability. The increase in the number of errors detected in the interval. The least squares method for approximating experimental data on software errors. Literature: [4 , 14, 17].	2
Topic 9. Modeling the work of CTS Legacy software. Software mobility and software reengineering.		2
17	**Lecture 17. Modeling the operation of the CTS in order to generate data for complex software debugging. Legacy software. Software mobility and software reengineering.** The problem of data generation for complex debugging. Principles of mathematical modeling. Advantages of mathematical modeling of the external environment. Legacy software. Software mobility and software reengineering. Emulation of the work of the old COM on the new hardware platform. References: [5, 7 , 16].	2
Total		**34**

5.2 Content of laboratory works

Number	Topics of laboratory classes	Hours
1	Software development planning. Stages of software development with a structural approach to programming. Stage "Technical task". Pre-project survey of the subject area, system analysis of the design object.	**2**
2	Structural approach to programming. "Sketch project" stage. Grid schedule of works. Selection of software development methodology. Description of software requirements. Work in a team.	**4**
3	Software development stage "Technical project". Structural approach to programming. Human-machine interface development . Development of a software prototype.	**4**
4	Stages of software development Stage "Implementation". Design and implementation of a software project with a user interface.	**4**
5	Project evaluation based on loc and fp metrics.	**4**
6	Sensitivity analysis of the software project based on the COCOMO II model	**6**
7	Metrics of object-oriented software systems.	**6**

8	Organization of tables in translators and work with them. Refactoring. Code optimization.	**4**
Total		**34**

5.3 Content of independent (individual) work

The volume of independent work in the discipline **"Software Design"** is 112 hours. They include study of lecture material, theoretical and laboratory tasks, preparation for laboratory works, their defense and current testing.

Number of the week	Topic name	Hours
1	**Topic 1. Subject and content of the discipline. The place of discipline in the structure of education. Software development problems and ways to solve them. Software engineering** . Elaboration of lecture material, preparation for laboratory work No. 1. Literature: [1-3; 7; 12].	6
2	**Topic 1. Subject and content of the discipline. The place of discipline in the structure of education. Software development problems and ways to solve them. Software engineering** . Processing of lecture material. Protection of laboratory work No. 1. Literature: [1-3; 7; 21].	6
3	**Topic 2. Systematic approach to software development. Software life cycle models. Software creation methodology.** Elaboration of lecture material, preparation for laboratory work No. 2. Literature: [2; 5; 13].	6
4	**Topic 2. Systematic approach to software development. Software life cycle models. Software creation methodology.** Processing of lecture material. Protection of laboratory work #2. Literature: [3-6 ; 11, 14 , 1 6].	6
5	**Topic 3. Designing when designing software.** Processing of lecture material. Preparation for laboratory work No. 3. Literature: [2, 4; 7; 8-10, 15 , 1 7].	6
6	**Topic 3. Designing when designing software.** Elaboration of lecture material, defense of laboratory work #3. Literature: [2; 4-6; 1 6 -1 7].	6
7	**Topic 4. Software design and multitasking.** Processing of lecture material. Preparation for laboratory work No. 4. Literature: [1; 4; 13].	6
8	**Topic 4. Software design and multitasking.** Elaboration of lecture material, defense of laboratory work #4. Literature: [1; 4; 13]. Preparation for testing.	6
9	**Topic 5. Protective software design, process synchronization, software design approaches (structural and OOP).** Elaboration of lecture material, preparation for laboratory work No. 5. Literature: [3-4; 6; 14]	6
10	**Topic 5. Protective software design, process synchronization, software design approaches (structural and OOP).** Elaboration of lecture material, defense of laboratory work No. 5. Literature: [3; 5; 7]	6
11	**Topic 6. Heuristic principles of program design and error protection.** Elaboration of lecture material, preparation for laboratory work No. 6. Literature: [1-2; 7; 12]	6
12	**Topic 6. Heuristic principles of program design and error protection.** Elaboration of lecture material, defense of laboratory work #6. Literature: [1-2; 7; 12].	6
13	**Topic 7. Error handling methods, software emergency protection. Software debugging technology. Software verification and validation methods** .	6

	Elaboration of lecture material, preparation for laboratory work No. 7 . Literature: [5-6, 14 , 1 6].	
14	**Topic 7. Error handling methods, software emergency protection. Software debugging technology. Software verification and validation methods** . Processing of lecture material. Protection of laboratory work #7. Literature: [3 , 4 , 16].	6
15	**Topic 8. Debugging during software design and their control. Instrumental software for domain analysis, design, testing, visualization, measurement, and software documentation.** Elaboration of lecture material, preparation for laboratory work No. 8. Literature: [4, 7, 12, 1 4 , 18].	6
16	**Topic 8. Debugging during software design and their control. Instrumental software for domain analysis, design, testing, visualization, measurement, and software documentation.** Elaboration of lecture material, performance of laboratory work No. 8. Literature: [4 , 13, 16 , 18].	6
17	**Topic 9. Modeling the work of STS Legacy software. Software mobility and software reengineering.** Elaboration of lecture material, defense of laboratory work #8. Literature: [5, 7 , 16]. Preparation for the exam	16
Total		**112**

6. TEACHING METHODS

The teaching process in the discipline is based on the use of traditional and modern methods: methods of problem-based teaching, verbal, visual (lectures); explanatory and illustrative, problem-based teaching, research, partially research-based (laboratory classes), problem-based teaching, research, partially research-based (independent work: individual tasks). All classes are held using information technologies and have the goal of acquiring practical skills in software design, testing and debugging by students of the first (bachelor) level of higher education, using metrics to evaluate the developed software.

7. ASSESSMENT FORMS AND METHODS

Current control is carried out during lectures and laboratory classes. Semester control is conducted in the form of an exam. When deriving the final grade, the results of the current control and the written exam are taken into account.

The process of assessing the readiness of an applicant for the first (bachelor) level of higher education can be divided into stages:

The first stage of assessment is aimed at determining the knowledge of the information minimum. If the student has firmly mastered the amount of formal knowledge determined by the curriculum, it means that he knows how to use it in solving various issues in the design of software systems, knows how to expand it.

Before studying a discipline, as a rule, there is an input control of knowledge from the disciplines that precede and provide it. At the same time, it is necessary to establish the levels and criteria of the formation of knowledge regarding the content of educational elements. These levels are:

Knowledgeable (OO) - a person has a rough understanding of the concepts being studied, is able to: program the main elements of software systems in various programming languages, choose modern software design methodologies and technologies, reasonably use modern software development environments for the development of software systems.

Conceptual-analytical (PA) - a person has a clear idea about the educational object, is able to transfer previously acquired knowledge to typical situations.

Productive-synthetic (PS) - a person has a deep understanding of the educational object, is able to carry out synthesis, generate new ideas and ideas, transfer previously acquired knowledge to atypical, non-standard situations.

Each type of work in the discipline is evaluated on *a four-point* scale. The semester final grade is defined as a weighted average of all types of academic work completed and passed *positively* , taking into account the weighting factor. The weighting factors change depending on the structure of the discipline and the importance of its individual types of work. An applicant who scored a positive weighted average score for the current work and did not pass the control measure (exam) is considered to have failed.

When assessing the knowledge of first (bachelor) level higher education students, various means of control are used, in particular: an oral survey before admission to the performance of laboratory work - it is carried out at the beginning of it; assimilation of theoretical material from topics is checked by test control; the quality of performance, acquisition of theoretical knowledge and practical skills is checked by defending each laboratory work in accordance with the work program of the discipline and the work curriculum.

The grade given for *the laboratory session* consists of the following elements: an oral interview of the test takers before admission to the laboratory work; knowledge of theoretical material on the topic; the quality of the design of the protocol and the graphic part; the acquirer's ability to justify the adopted constructive decisions; timely protection of laboratory work. To complete the discipline program, the applicant must obtain 7 grades for laboratory work.

The deadline for the defense of laboratory work is considered timely if the applicant defended it at the next class after the completion of the work.

The student must complete the missed laboratory class no later than two weeks before the end of theoretical classes in the semester.

When *evaluating the knowledge* of first (bachelor) level higher education students, the teacher is guided by the following criteria.

The student receives an "excellent" grade for deep and complete mastery of the content of the educational material, in which he can easily navigate, conceptual apparatus, for the ability to connect theory with practice, solve practical tasks, express and justify his judgments. An excellent assessment implies a competent, logical presentation of the answer (both orally and in writing), high-quality external design. The applicant must acquire practical skills in the design and software implementation of software systems.

The grade "excellent" is awarded to the applicant who has thoroughly mastered the basic principles of designing software systems and is able to apply them rationally, knows the methods and knows how to use them in the development of software. The applicant should not hesitate when changing the question, should make detailed and general conclusions.

The applicant receives a grade of "good" for complete assimilation of the educational material, mastery of the conceptual apparatus, orientation in the studied material, conscious use of knowledge to solve practical tasks, competent presentation of the answer, but there were some inaccuracies (errors) in the content and form of the answer, unclear formulations of regularities etc. The applicant's answer should be based on independent thinking.

The winner receives a "good" grade for a correct answer with one or two significant errors.

The grade "satisfactory" is deserved by the applicant who has demonstrated knowledge of the main educational and program material in the amount necessary for further training and practical activity in the profession, which copes with the implementation of practical tasks provided for by the program. As a rule, the applicant's answer is built on the level of reproductive thinking, the applicant has little knowledge of the structure of the course, makes mistakes in the answer, learned and acquired practical skills in the design and implementation of software systems, but made inaccuracies. Hesitates when answering a modified question, at the same time, the applicant has knowledge that allows him to eliminate inaccuracies in the answer under the guidance of the teacher.

The winner deserves a "satisfactory" grade for incomplete mastery of software material, but acquired knowledge and acquired practical skills in software design and development.

The grade "unsatisfactory" is assigned when the student has scattered, unsystematic knowledge, does not know how to distinguish the main from the secondary, makes mistakes in defining concepts,

distorts their meaning, presents the material chaotically and uncertainly, cannot use knowledge when solving practical tasks.

Based on the results of the current control and exam, a final semester grade is issued.

Credit is issued when the student receives from 3.00 to 5.00 points in the discipline. At the same time, according to the national scale, "credited" is given, and according to the ECTS scale, the number of points scored by the student and the corresponding grade.

When teaching the discipline, such types of training are used as lectures, laboratory work, individual counseling and guidance of the applicant's independent work, including by individual task.

When evaluating the knowledge of applicants, various control tools are used, in particular: admission to the performance of laboratory work is carried out at its beginning by an oral interview of each applicant; assimilation of the theoretical material of content modules is checked by content control; the quality of performance, acquisition of theoretical knowledge and practical skills is checked by defending each laboratory work and individual task according to the work plan.

The grade given for the laboratory session consists of the following elements: an oral interview of the test takers before admission to the laboratory work; knowledge of theoretical material on the topic; the quality of the design of the protocol and the graphic part; the acquirer's ability to justify the adopted constructive decisions; timely protection of laboratory work.

The deadline for the defense of laboratory work is considered timely if the applicant defended it at the next class after the completion of the work.

A candidate who missed a laboratory session for a good reason must complete it in the department's laboratories within the time limit set by the teacher.

Structuring of the discipline by types of work and evaluation of the study results of applicants in the semester by weighting coefficients

<table>
<tr><td colspan="8">Auditory work</td><td rowspan="2">Semester control, exam (I)</td><td rowspan="2">Final score</td></tr>
<tr><td colspan="7">Laboratory work (LR)</td><td>Test</td></tr>
<tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>T</td><td rowspan="2">0.4</td><td rowspan="2">LR*0.3+T*0.3+I*0.4</td></tr>
<tr><td colspan="7">VC = 0.3</td><td>VC = 0.3</td></tr>
</table>

Conventional designations: VC - weighting factor, LR - laboratory work, T - test, I - exam.

Evaluation of test tasks. The thematic test for each applicant consists of twenty test tasks, each of which is evaluated by one point. The maximum amount of points that the winner can score is 20.

Evaluation is carried out on a four-point scale.

Correspondence of the scored points for the test task to the grade given to the applicant is presented in the table below.

The sum of points for the test task	1–11	12–14	15–18	19-20
Rating	2	3	4	5

20 minutes are allotted for testing. Testing is conducted using the MOODLE modular learning environment. The winner registers the correct answers online in the MOODLE modular environment. After 30 minutes, test takers complete the test and send their answers to the server. The teacher announces the test results according to the evaluation log of the MOODLE modular environment.

If the applicant received a negative grade, he must resubmit it in accordance with the established procedure, but necessarily before the next control period.

The final semester grade according to the national scale and the ECTS scale is set in an automated mode after entering all grades into the electronic journal. The ratio of the domestic assessment scale and the ECTS assessment scale is shown in the following table.

Correlation of the domestic evaluation scale and the ECTS evaluation scale

Evaluation of ECTS	Institutional interval scoring scale	Domestic assessment, criteria		
A	4.75–5.00	5	***Excellent*** - deep and complete mastery of the educational material and identification of relevant abilities and skills	Enrolled
B	4.25–4.74	4	***Good*** - complete knowledge of the educational material with a few minor errors	
C	3.75–4.24	4	***Good*** - a generally correct answer with two or three significant errors	
D	3.25–3.74	3	***Satisfactory*** - incomplete mastery of the program material, but sufficient for practical activities in the profession	
E	3.00–3.24	3	***Satisfactory*** - incomplete mastery of the program material that meets the minimum evaluation criteria	
FX	2.00–2.99	2	***Unsatisfactory*** – unsystematic knowledge acquired and the impossibility of continuing education without additional knowledge of the discipline	Not counted
F	0.00–1.99	2	***Unsatisfactory*** - serious further work and re-study of the discipline is necessary	

Credit is given if the average weighted score received by the student in the discipline is between 3.00 and 5.00 points. At the same time, according to the national scale, the grade "passed" is given , and according to the ECTS scale, the letter designation of the grade corresponds to the number of points scored by the applicant according to the Ratio table.

8. QUESTIONS FOR STUDENTS' SELF-CONTROL

1. Cascade models of the software development life cycle
2. Package diagrams in UML
3. Spiral models of the software development life cycle
4. Ways of setting constraints in UML diagrams.
5. Spiral models of the development life cycle of object-oriented software systems
6. Types of relationships used in class diagrams
7. Work carried out at the stage of analysis for the life cycle of the development of software systems.
8. Methods of displaying active objects in UML diagrams.
9. Work performed during the specification of requirements for software systems.
10. Inheritance relationships in class diagrams.
11. Works performed at the system analysis stage
12. Aggregation relations in class diagrams.
13. Works performed at the object analysis stage.
14. Association relations in class diagrams and methods of their implementation in object-oriented programming languages
15. Works performed at the design stage.
16. Ways of representing multi-layered architectures using UML tools.
17. Works performed at the mechanism design stage.
18. Similarities and differences between automatic models and Harel diagrams used in state diagrams in UML.
19. Works performed at the stage of detailed design.
20. Parameterization of classes in UML.
21. Works performed at the architectural design stage.
22. Types of constraints in embedded and real-time systems.
23. Work performed at the coding stage.

24. The use of UML-diagrams at various stages of the software system development life cycle.
25. Works performed at the testing stage.
26. Software requirements specification languages.
27. Case diagrams (UseCase diagrams) in UML.
28. The concept of class purpose.
29. Object diagrams (interaction diagrams) in UML.
30. Relationship types in UseCase diagrams.
31. Class diagrams in UML.
32. Design tools that use UML as an input language.
33. State diagrams in UML.
34. Scenarios for UseCase charts and how to present them.
35. Sequence diagrams in UML.
36. Association relation and its use in program generation.
37. Deployment diagrams in UML.
38. Ways to set constraints in sequence diagrams in UML.
39. Component diagrams in UML.
40. Class diagrams and their role in deployment diagrams.
41. evaluation based on loc- and fp-metrics.
42. Sensitivity analysis of the software project based on the COCOMO II model
43. The principle of refactoring.
44. Metrics of object-oriented software systems.
45. Code optimization, principle of operation.

9. TEACHING AND LEARNING MATERIALS

The educational process in the discipline is fully and in sufficient quantity provided with the necessary educational and methodical literature.

10. RECOMMENDED LITERATURE

MAIN

1. Роберт Мартін. Чистий код: Чистий код: створення, аналіз, рефакторинг. - Фабула. 2019 – 416 с.
2. Постіл. С.Д. UML. Уніфікована мова моделювання інформаційних систем: Навч. посіб., 2019. - 321 с.
3. Piotr Sliż. Organizacja procesowo-projektowa. Istota, modelowanie, pomiar dojrzałości. Difin – 2021 – 292 str.
4. Ian Sommerville. Software Engineering, 10th edition. Published by Pearson . July 14th 2021 – 816 p.
5. Philippe Kruchten. The Rational Unified Process: An Introduction (5rd Edition) (Addison-wesley Object Technology Series). Addison-Wesley Professional; 5rd edition. 2018 – 407 p.
6. Pinto J. Project Management: Achieving Competitive Advantage, 5th Edition. – 2019. – 592 p.
7. Martin Fowler. Refactoring: ImprovingtheDesignofExistingCode (WebEdition), 2nd Edition/ WebEdition.- 2018

AUXILIARY:

8. Shyam R Chidamber,Chris F Kemerer (Author), Sloan School of Management. A Metrics Suite for Object Oriented Design. Franklin Classics. - 2018 – 44 p.
9. D.Spinellis.Programcodeanalysis -AddisonWesley, 2004
10. ErichGamma, RichardHelm, RalphJohnson, andJohnVlissides. DesignPatterns: ElementsofReusable Object-Oriented Software.Addison-Wesley - 2009 - 417p.

11. Beck Kent. Extreme Programming Explained: EmbraceChange (The XP Series). Addison-WesleyProfessional. – 2004 - 224
12. Dave Nicolette. Software Development Metrics 1st Edition - Manning; 1st edition – 2015 – 192 p.
13. Eric J. Braude; Michael E. Bernstein. Software Engineering. Waveland Press, 2016 – 802 p.
14. Bohm, Corrado; andGiuseppeJacopini (May 1966). "FlowDiagrams, TuringMachinesandLanguageswithOnlyTwoFormationRules". Communicationsofthe ACM 9 (5): 366–371. doi:10.1145/355592.365646
15. Dijkstra, E. W. (Aug 1972). "The Humble Programmer". Communications of the ACM 15 (10): 859–866. doi:10.1145/355604.361591.
16. http://www.cs.utexas.edu/~EWD/transcriptions/EWD03xx/EWD340.html. (EWD340) PDF, 1972 ACM TuringAwardlecture
17. RussellGold, ThomasHammell, TomSnyder. TestDrivenDevelopment: A J2EE Example.- Apress, 2005.- 296 pages.
18. Software design, architecture and engineering: concepts and practice. (2021). (n.p.): PHI Learning Pvt. Ltd.. - 280p.
19. Voorhees, D. P. (2021). Guide to Efficient Software Design: An MVC Approach to Concepts, Structures, and Models. Springer International Publishing. - 519p.
20. Sufyan bin Uzayr, Software Design Patterns: The Ultimate Guide (2021) CRC Press, - 454 p
21. Hanson, C., Sussman, G. J. (2021). Software Design for Flexibility: How to Avoid Programming Yourself Into a Corner. MIT Press. - 519p.
22. Parikh, K., Johri, A. (2022). Combining DataOps, MLOps and DevOps: Outperform Analytics and Software Development with Expert Practices on Process Optimization and Automation (English Edition). BPB Publications. – 400p
23. Bhatt, P. C. P. (2020). Software Design: Concepts and Practice. (n.p.): Independently Published. – 318p
24. Beningo, J. (2022). Embedded Software Design: A Practical Approach to Architecture, Processes, and Coding Techniques. : Apress. . – 463p
25. Chatterjee, S., Deshpande, S., Been, H., Gaag, M. v. d. (2022). Designing and Implementing Microsoft DevOps Solutions AZ-400 Exam Guide: Prepare for the Certification Exam and Successfully Apply Azure DevOps Strategies with Practical Labs. Packt Publishing. – 490p
26. Chin, S., McKay, M., Ruiz, I., Sadogursky, B. (2022). DevOps Tools for Java Developers: Best Practices from Source Code to Production Containers. O'Reilly Media, Incorporated. – 322 p
27. Праворська Н.І., Яшина О.М., Нетреба І.В. Доміна А.Р. Кириченко О.М. Метод конструювання програмного забезпечення згідно аналізу помилок SQL-запитів. Вісник Хмельницького національного університету, серія Технічні науки. №3, 2023 (321) – с. 302-307
28. Праворська Н.І., Мартинюк В.В. Конструювання програмного забезпечення за допомогою синхронного підходу: основні процеси та інструменти для ефективної реалізації devops. Вісник Хмельницького національного університету, Том 1, №5, 2023 (325) – стор.182-192

11. INFORMATION RESOURCES

1. MOODLE Learning Platform. Access to the resource https://msn.khmnu.edu.ua.
2. University Electronic Library. Access to the resource: http://library.khmnu.edu.ua.
3. University Repository. Access to the resource https://elar.khmnu.edu.ua/home.