МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Тернопільський національний технічний університет імені Івана Пулюя

Кафедра програмної інженерії

Методичні вказівки для самостійної роботи

з дисципліни

«Раціональний уніфікований процес проєктування програмного забезпечення»

для здобувачів другого (магістерського) рівня вищої освіти
ОПП «Інженерія програмного забезпечення»
всіх форм навчання

Тернопіль — 2026

УДК 004.41(076.5)

Укладачі:
І.Я. Мудрик, PhD

Рецензент:
І.О. Боднарчук, к.т.н., доцент
завідувач кафедри комп'ютерних наук ТНТУ ім. І. Пулюя

Розглянуто й затверджено на засіданні кафедри програмної інженерії.
Протокол № 12 від 30.03.2026.

Розглянуто й затверджено на засіданні методичної комісії факультету комп'ютерно-інформаційних систем та програмної інженерії Тернопільського національного технічного університету імені Івана Пулюя.
Протокол № 4 від 09.04.2026.

Методичні вказівки для самостійної роботи з дисципліни «Раціональний уніфікований процес проєктування програмного забезпечення» для здобувачів другого (магістерського) рівня вищої освіти ОПП «Інженерія програмного забезпечення» всіх форм навчання / Укладач: Мудрик І.Я.,– Тернопіль: ТНТУ ім. І. Пулюя, 2026 – 33 с.

Відповідальний за випуск: І.Я. Мудрик, PhD

АНОТАЦІЯ

Самостійна робота студента (СРС) є невід'ємною складовою освітнього процесу, яка має на меті поглиблення, узагальнення та закріплення теоретичних знань, отриманих під час лекційних занять, а також підготовку до виконання лабораторних робіт.

У межах вивчення дисципліни «Раціональний уніфікований процес проєктування ПЗ» самостійна робота спрямована на формування у майбутніх інженерів програмного забезпечення навичок самостійного аналізу архітектурних патернів, дослідження сучасної проєктної документації, а також вивчення інтеграції класичних ітеративних процесів із сучасними практиками (DevOps, Agile, AI Engineering).

Основні види самостійної роботи:

- Опрацювання лекційного матеріалу та рекомендованої літератури.
- Підготовка до виконання та захисту лабораторних робіт (встановлення та налаштування CASE-засобів, вивчення нотації UML).
- Написання рефератів, есе або підготовка презентацій за заданою тематикою.
- Самостійне вивчення тем, які не увійшли до лекційного курсу, але передбачені робочою програмою.
- Підготовка до модульного та підсумкового контролю.

ЗМІСТ

ВСТУП

Сучасна індустрія інженерії програмного забезпечення розвивається надзвичайно стрімко. Технології, фреймворки та мови програмування постійно оновлюються, проте фундаментальні принципи проєктування надійних, масштабованих та зручних у супроводі систем залишаються незмінними. Дисципліна «Раціональний уніфікований процес проєктування ПЗ» (RUP) покликана сформувати у майбутніх інженерів саме таке фундаментальне архітектурне мислення, навчити керувати вимогами, управляти ризиками та перетворювати хаос бізнес-ідей у чітко структуровані візуальні моделі та програмний код.

В умовах кредитно-модульної системи організації навчального процесу **самостійна робота студента (СРС)** є не просто додатковим навантаженням, а основним засобом оволодіння навчальним матеріалом у час, вільний від обов’язкових аудиторних занять. Аудиторні лекції та лабораторні роботи дають базовий інструментарій і задають напрямок, тоді як глибоке розуміння предмета приходить лише через самостійне дослідження.

Головна мета самостійної роботи з цієї дисципліни — навчити студента самостійно мислити категоріями об'єктно-орієнтованого аналізу, знаходити оптимальні архітектурні рішення, критично оцінювати існуючі патерни та адаптувати класичні ітеративні методології розробки до сучасних практик (Agile, DevOps, хмарні технології).

У процесі самостійної роботи студенти повинні:

- поглибити теоретичні знання щодо життєвого циклу розробки ПЗ;
- досконало опанувати синтаксис та семантику уніфікованої мови моделювання (UML) для створення проєктних артефактів;
- навчитися самостійно працювати із сучасною англомовною технічною літературою, специфікаціями (наприклад, OMG UML) та міжнародними стандартами інженерії ПЗ;
- опанувати інструментарій CASE-засобів візуального моделювання для підготовки до захисту лабораторних проєктів;
- дослідити сучасні тренди в архітектурі (мікросервіси, предметно-орієнтоване проєктування, використання ШІ-асистентів у розробці).

Усі матеріали, необхідні для виконання завдань самостійної роботи (теми для досліджень, вимоги до рефератів і презентацій, питання для самоконтролю), а

також терміни їх здачі узгоджуються з викладачем та розміщуються у системі електронного навчання курсу.

Враховуючи сучасні тенденції розвитку інженерії програмного забезпечення, ці вказівки побудовані так, щоб студенти не лише вивчали класичний RUP, але й критично аналізували його в контексті сучасних підходів (Agile, DevOps) та використання новітніх інструментів (зокрема AI-асистентів).

Варто пам'ятати, що інженерія програмного забезпечення — це галузь, яка вимагає найвищої професійної етики. Тому всі види самостійної роботи повинні виконуватися з суворим дотриманням принципів академічної доброчесності (відповідно до Положення про академічну доброчесність учасників освітнього процесу Тернопільського національного технічного університету імені Івана Пулюя). Оригінальність архітектурних рішень, самостійність побудови моделей та неприпустимість плагіату є ключовими критеріями успішного засвоєння дисципліни.

ТЕМАТИЧНИЙ ПЛАН НАВЧАЛЬНОЇ ДИСЦИПЛІНИ

Дисципліна: Раціональний уніфікований процес проєктування програмного забезпечення.

Рівень: другий (освітньо-професійний).

Спеціальність: «Інженерія програмного забезпечення»

Форма контролю (двосеместрово): залік та екзамен.

Обсяг: 8.0 ECTS (240 годин): 84 год. аудиторних (48 лекції, 48 практичні), 156 год. самостійної роботи (з них – виконання курсових проектів24 год.).

Метою та завданням курсу є розвинення у студентів навичок та підходів щодо використання перевірених архітектурних принципів і шаблонів проектування для створення ефективних високоякісних програмних комплексів в більш стислі терміни і з меншими ризиками. Можливість формування у студентів комплексу знань та практичних навичок із планування, моніторингу та управлінню проектами від локального до корпоративного рівнів з використанням глобальної методології РУП. Розглядаються принципи розробки інформаційних систем, структурної організації команди розробників системи, їх взаємодія в процесі функціонування, перспективні напрямки розвитку сучасних інформаційних технологій, із вдосконалення навичок використання методології RUP. Це направлено на ітеративну модель розробки, а після закінчення кожного етапу і кожної ітерації створювати всі необхідні документи і досягти максимального рівня формалізації.

Мета проведення лекційних занять полягає в ознайомленні студентів з теоретичними основами проектування ПЗ та архітектурних рішень, ознайомлення з шаблонами проектування, середовищами проєктування архітектур та моделей. Дослідження існуючих архітектур проміжного програмного забезпечення згідно основних підходів RUP.

Мета проведення лабораторних занять полягає у тому, щоб виробити у студентів практичні навички застосування уніфікованих методів та підходів до проектування ПЗ згідно загальноприйнятих правил методології RUP та розробки архітектури програмного забезпечення у відповідності до проєктованих рішень.

При цьому завданням курсу є ознайомити студентів з процесом організації проектування та розробки ПЗ, розвинути навики організації праці в команді, допомогти майбутнім фахівцям оволодіти знаннями менеджменту розробки ПЗ з погляду процесу RUP.

2.2. За результатами вивчення дисципліни «Раціональний уніфікований процес проектування програмного забезпечення» студент повинен

продемонструвати такі результати навчання (***кінцеві, підсумкові та інтегративні результати навчання, що визначають нормативний зміст підготовки):***

За результатами вивчення дисципліни «Раціональний уніфікований процес проектування програмного забезпечення» студент повинен продемонструвати такі **результати навчання**:

РН01 Знати і застосовувати сучасні професійні стандарти і інші нормативно-правові документи з інженерії програмного забезпечення

РН02 Оцінювати і вибирати ефективні методи і моделі розроблення, впровадження, супроводу програмного забезпечення та управління відповідними процесами на всіх етапах життєвого циклу.

РН03 Будувати і досліджувати моделі інформаційних процесів у прикладній області.

РН06 Розробляти і оцінювати стратегії проєктування програмних засобів; обґрунтовувати, аналізувати і оцінювати варіанти проектних рішень з точки зору якості кінцевого програмного продукту, ресурсних обмежень та інших факторів.

РН08 Розробляти і модифікувати архітектуру програмного забезпечення для реалізації вимог замовника.

РН11 Забезпечувати якість на всіх стадіях життєвого циклу програмного забезпечення, у тому числі з використанням релевантних моделей та методів оцінювання, а також засобів автоматизованого тестування і верифікації програмного забезпечення.

РН17 Збирати, аналізувати, оцінювати необхідну для розв’язання наукових і прикладних задач інформацію, використовуючи науково-технічну літературу, бази даних та інші джерела.

Вивчення навчальної дисципліни передбачає формування та розвиток у студентів **компетентностей**:

інтегральна компетентність:

- Здатність особи розв’язувати складні задачі і проблеми у певній галузі професійної діяльності або у процесі навчання, що передбачає проведення досліджень та/або здійснення інновацій та характеризується невизначеністю умов і вимог.

загальні компетентності:

- ЗК01. Здатність до абстрактного мислення, аналізу та синтезу
- ЗК05. Здатність генерувати нові ідеї (креативність). ЗК05. Здатність генерувати нові ідеї (креативність).

спеціальні (фахові) компетентності:

- СК01. Здатність аналізувати предметні області, формувати, класифікувати вимоги до програмного забезпечення.
- СК02. Здатність розробляти і реалізовувати наукові та/або прикладні проєкти у сфері інженерії програмного забезпечення.
- СК03. Здатність проєктувати архітектуру програмного забезпечення, моделювати процеси функціонування окремих підсистем і модулів.
- СК05. Здатність розробляти, аналізувати та застосовувати специфікації, стандарти, правила і рекомендації в сфері інженерії програмного забезпечення.
- СК09. Здатність забезпечувати якість програмного забезпечення.
- СК10*. Здатність розробляти інноваційне програмне забезпечення для систем підтримки прийняття рішень.
- СК* - спеціальна компетентність, запропонована стейкхолдерами.

Знання: Зн1. Спеціалізовані концептуальні знання, що включають сучасні наукові здобутки у сфері професійної діяльності або галузі знань і є основою для оригінального мислення та проведення досліджень;

Уміння: Ум1, Ум2. Спеціалізовані уміння/навички розв’язання проблем, необхідні для проведення досліджень та/або провадження інноваційної діяльності з метою розвитку нових знань та процедур. Здатність інтегрувати знання та розв’язувати складні задачі у широких або мультидисциплінарних контекстах;

Комунікація: К1. Зрозуміле і недвозначне донесення власних знань, висновків та аргументації до фахівців і нефахівців, зокрема, до осіб, які навчаються;

Автономія та відповідальність: АВ1. Управління робочими або навчальними процесами, які є складними, непередбачуваними та потребують нових стратегічних підходів.

ВИКОНАННЯ КУРСОВОГО ПРОЄКТУ

Після курсу лекцій у першому семестрі студенти виконують курсовий проєкт (КП).

Метою курсового проекту (КП) є закріплення знань, отриманих на лекційних та лабораторних заняттях, набуття навиків практичного їх використання, а також досвіду складання пояснювальних записок, обґрунтування своїх рішень, що приймаються на підставі отримуваної у висліді опрацювання даних інформації.

Проєкт виконується магістрантом самостійно протягом семестру і є обов'язковою умовою допуску до підсумкового контролю.

Суть завдання: Застосування адаптованого процесу RUP для проєктування архітектури розподіленої програмної системи (тема обирається студентом за погодженням з викладачем або відповідає темі магістерської роботи).

Структура звіту з КП:

1. Vision Document (Бачення): Опис проблеми, ключових стейкхолдерів, бізнес-вимог та обмежень.
2. Risk Assessment (Оцінка ризиків): Ідентифікація мінімум 10 ризиків та план їх мінімізації.
3. Software Architecture Document (SAD):

- Логічне представлення (Logical View) – діаграми класів, пакетів.
- Представлення процесів (Process View) – діаграми активності або послідовності.
- Представлення розгортання (Deployment View) – діаграма розгортання, опис інфраструктури (напр., хмарне розгортання, GCP/AWS).

4. Аналітична записка: Обґрунтування того, як саме використання ітерацій RUP допомогло вирішити виявлені архітектурні ризики, та короткий висновок щодо доцільності застосування DevOps-практик для подальшої підтримки створеної системи.

Завданням КП з предмету є ознайомити студентів з процесом організації проектування та розробки ПЗ згідно повного циклу виконання проєктів розробки, розвинути навики організації праці в команді, допомогти фахівцям оволодіти знаннями менеджменту розробки ПЗ з погляду методології виконання RUP, направлену на ітеративну модель розробки, а після закінчення кожного етапу і кожної ітерації створювати всі необхідні документи і досягти максимального рівня формалізації. Поглибити, закріпити знання, отримані при вивченні даної дисципліни, розвинути здатність до творчої і самостійної роботи, навчити використовувати знання при розв’язуванні конкретних задач.

Зміст КП у частині, що стосується цієї дисципліни — використання підходу РУП з метою організації повного та стабільного підходу до виконання проєкту

згідно заданої методології розробки ПЗ. Під час виконанні КР студенти практично використовують комп’ютерні засоби та технології виконання замкненого процесу розробки ПЗ згідно життєвого циклу, та технології, навчальну та наукову електронну апаратуру.

З тематикою КП пов’язані теми лабораторних. Основою для виконання КП є знання отримані в процесі опрацювання лекційного матеріалу, лабораторних занять, самостійної роботи.

В методичних вказівках наведено варіанти типових завдань і рекомендації по виконанню та оформленню КП.

У методичних вказівках приведено варіанти типових завдань і рекомендації по виконанню та оформлення записки до КП.

Теми КП враховують тематики науково-дослідних робіт кафедри програмної інженерії.

Розподіл балів, які отримають студенти при виконанні КП:

Пояснювальна записка	Ілюстративна частина	Захист роботи	Сума
до 50 балів	до 25 балів	до 25 балів	100

ВИМОГИ ДО ОФОРМЛЕННЯ МАТЕРІАЛІВ

- Звітна документація (КП) оформлюється у вигляді текстового документа формату А4 (шрифт Times New Roman, 14 пт, інтервал 1.5) з дотриманням стандартів ДСТУ щодо оформлення студентських робіт.
- Усі UML-діаграми повинні мати чітку роздільну здатність, підписи та супровідний текст з поясненням елементів.
- Рекомендується використання сучасних інструментів моделювання (Sparx Enterprise Architect, Visual Paradigm, Lucidchart або PlantUML) з обов'язковим додаванням посилань на вихідні файли або репозиторій.

КРИТЕРІЇ ОЦІНЮВАННЯ РЕЗУЛЬТАТІВ НАВЧАННЯ СТУДЕНТІВ

Теоретичний курс (тестування): Оцінюється глибина розуміння теоретичного матеріалу, знання основних концепцій, методів аналізу та моделювання.

Лабораторні роботи: Оцінюється якість виконання лабораторних завдань, прикладне застосування теоретичних знань, коректність результатів та обґрунтованість висновків.

Форма підсумкового семестрового контролю оцінювання студентів за IX семестр: – залік, курсовий проект.

Модуль 1			Модуль 2			Підсумковий контроль	Разом з дисципліни
Аудиторна та самостійна робота			Аудиторна та самостійна робота				
Теоретичний курс (тестування)	Практична робота		Теоретичний курс (тестування)	Практична робота			
15	**20**		**20**	**20**		**25**	**100**
№ лекцій	Вид робіт	Бал	№ лекцій	Вид робіт	Бал	за кожних три бали семестрової оцінки студент отримує 1 бал підсумкової семестрової оцінки автоматично	
Вступна	Техніка безпеки		Лекція № 7				
Лекція № 1			Лекція № 8	Лаб. роб. № 3	10		
Лекція № 2			Лекція № 9				
Лекція № 3	Лаб. роб. № 1	10	Лекція № 10				
Лекція № 4			Лекція № 11	Лаб. роб. № 4	10		
Лекція № 5			Лекція № 12				
Лекція № 6	Лаб. роб. № 2	10	Лекція № 13				
Проміжний кон-ль			Підсумковий кон-ль				

Форма підсумкового семестрового контролю оцінювання студентів за X семестр: – екзамен.

Модуль 3			Модуль 4			Підсумковий контроль	Разом з дисципліни
Аудиторна та самостійна робота			Аудиторна та самостійна робота				
Теоретичний курс (тестування)	Практична робота		Теоретичний курс (тестування)	Практична робота			
15	**20**		**15**	**25**		**25**	**100**
№ лекцій	Вид робіт	Бал	№ лекцій	Вид робіт	Бал	Підсумковий екзаменаційний Тест (до 15 балів) Практичне завдання (до 10 балів)	
Вступна	Техніка безпеки		Лекція № 22		6		
Лекція № 16			Лекція № 23	Лаб. роб.№ 7	10		
Лекція № 17	Лаб. роб.№ 5	10	Лекція № 24				
Лекція № 18			Лекція № 25				
Лекція № 19			Лекція № 26	Лаб. роб.№ 8 проєкт	15		
Лекція № 20	Лаб. роб.№ 6	10	Лекція № 27				
Лекція № 21			Лекція № 28				
Проміжний кон-ль			Підсумковий кон-ль				

Оцінювання результатів СРС здійснюється під час перевірки звітів з лабораторних робіт, виступів з рефератами (презентаціями) та під час поточного модульного контролю. Оцінювання самостійної роботи здійснюється за 100-бальною шкалою (з подальшим переведенням у національну шкалу та шкалу ECTS) і враховує:

- **Глибину опрацювання матеріалу (до 30 балів):** Повнота відповідей на питання для самоконтролю під час усного опитування або тестування. Здатність студента аргументовано порівнювати методології (RUP vs Agile/DevOps).
- **Якість виконання ІНДЗ (до 50 балів):**
  - Коректність використання нотації UML (15 балів).
  - Повнота та логічність архітектурного документа SAD (20 балів).
  - Реалістичність оцінки ризиків (15 балів).
- **Захист індивідуального проєкту (до 20 балів):** Здатність магістранта захистити прийняті архітектурні рішення, продемонструвати розуміння фаз RUP та відповісти на додаткові питання.

При оцінюванні враховується:

- Самостійність виконання (відсутність плагіату).
- Глибина розкриття теми та здатність аргументувати вибір архітектурних рішень.
- Коректність використання термінології програмної інженерії.
- Відповідність розроблених UML-моделей офіційній специфікації.
- Вміння пов'язувати класичну теорію із сучасними технологічними стеками та інженерними практиками.

Шкала офінювання:

Сума балів за навчальну діяльність	Оцінка ECTS	Оцінка за національною шкалою
90 – 100	A	Відмінно
82–89	B	Добре
74–81	C	Добре
64–73	D	Задовільно
60–63	E	Задовільно
35–59	FX	Незадовільно з можливістю повторного складання
0–34	F	Незадовільно з обов'язковим повторним вивченням дисципліни

Оцінка «A» (90–100 балів):

- Глибоке розуміння фундаментальних принципів Раціонального уніфікованого процесу (RUP), архітектурно-орієнтованого підходу та ітеративного життєвого циклу розробки.
- Безпомилково виконані всі практичні завдання та ІНДЗ, включаючи детальну розробку моделі «4+1» та специфікацій вимог.
- Логічне та обґрунтоване розв’язання складних архітектурних задач, ефективне управління проєктними ризиками на ранніх етапах.
- Демонстрація аналітичних здібностей та творчого підходу до адаптації практик RUP у контексті сучасних DevOps та Agile методологій.

Оцінка «B» (82–89 балів):

- Добре розуміння фаз та дисциплін RUP, принципів управління вимогами та моделювання програмних систем.
- Якісне виконання практичних робіт зі створення UML-діаграм та проєктної документації (Vision, Risk List, SAD) з незначними помилками.
- Коректне застосування патернів проєктування та методів трасування вимог.
- Достатня глибина та точність в обґрунтуванні вибору архітектурних рішень для розподілених систем.

Оцінка «C» (74–81 балів):

- Задовільне розуміння основних тем курсу (фази Inception, Elaboration, Construction, Transition; ключові артефакти).
- Виконання практичних робіт з візуального моделювання (UML) з деякими процедурними чи семантичними помилками.
- Розв’язання стандартних задач з проєктування ПЗ без істотних проблем.
- Задовільне обґрунтування прийнятих рішень щодо структури програмного забезпечення.

Оцінка «D–E» (60–73 балів):

- Мінімальне розуміння основних концепцій та ролей у методології RUP.
- Практичні роботи (моделювання прецедентів, проєктування класів) виконані зі значними прогалинами або відсутністю ключових артефактів.
- Розв’язання простих задач з проєктування супроводжується помилками в нотації чи логіці.
- Слабке обґрунтування відповідей під час захисту архітектурних рішень.

Оцінка «FX» (35–59 балів):

- Неповне розуміння теоретичного матеріалу щодо процесів життєвого циклу ПЗ.
- Суттєві недоліки у виконанні практичних робіт (наприклад, грубе порушення стандартів UML, невідповідність архітектури поставленим вимогам).
- Помилки при розв’язанні базових задач з аналізу та проєктування.
- Можливість повторного складання або доопрацювання проєктної документації (ІНДЗ).

Оцінка «F» (0–34 балів):

- Неадекватне розуміння основних концепцій інженерії програмного забезпечення та методології RUP.
- Неякісне виконання або повна відсутність практичних робіт та індивідуального завдання.
- Неспроможність розв’язати навіть прості задачі з моделювання чи управління ризиками.
- Вимагає обов’язкового повторного вивчення дисципліни.

ТЕМАТИКА ТА ЗМІСТ САМОСТІЙНОЇ РОБОТИ СТУДЕНТІВ (СРС)

Самостійна робота магістрантів передбачає поглиблене опрацювання теоретичного матеріалу, підготовку до практичних занять та виконання індивідуального проєкту. Нижче наведено перелік тем для самостійного вивчення з відповідними завданнями.

Тема 1. Еволюція методологій: RUP у сучасному ландшафті розробки

Мета: Зрозуміти місце важковагових процесів у сучасній розробці ПЗ та провести їх порівняльний аналіз із гнучкими методологіями.

- **Завдання для самостійного опрацювання:**
    1. Провести порівняльний аналіз життєвого циклу RUP (Inception, Elaboration, Construction, Transition) із циклами розробки в Scrum та Kanban.
    2. Дослідити можливості інтеграції практик RUP із сучасними DevOps-пайплайнами (Continuous Integration / Continuous Deployment).
    3. Проаналізувати концепцію "Agile RUP" (AUP) та OpenUP.
- **Питання для самоконтролю:**
    1. Які дисципліни RUP найскладніше адаптувати до швидких релізів (DevOps)?
    2. В яких типах проєктів використання повного циклу RUP залишається економічно та технічно обґрунтованим?

Тема 2. Архітектурне моделювання та управління вимогами

Мета: Опанувати практики управління складними вимогами та проєктування надійної архітектури (Architecture-centric approach).

- **Завдання для самостійного опрацювання:**
    1. Створити модель прецедентів (Use Case Model) для гіпотетичної високонавантаженої системи (наприклад, платформи агрегації новин або IoT-системи енергоменеджменту).
    2. Побудувати архітектурне представлення "4+1 View Model" Філіпа Кручтена з використанням UML 2.x.
    3. Опрацювати методи специфікації нефункціональних вимог (FURPS+).
- **Питання для самоконтролю:**
    1. Як модель "4+1" допомагає вирішувати конфлікти між зацікавленими сторонами (Stakeholders)?
    2. Яка роль архітектурного прототипу на фазі Elaboration?

Тема 3. Управління ризиками та ітераційне планування

Мета: Навчитися ідентифікувати, оцінювати та пом'якшувати ризики на ранніх етапах життєвого циклу.

- **Завдання для самостійного опрацювання:**
    1. Розробити реєстр ризиків (Risk List) для індивідуального проєкту.
    2. Описати стратегії пом'якшення (Mitigation strategies) для топ-5 архітектурних та технологічних ризиків.
    3. Скласти план ітерацій для фази Construction, спираючись на пріоритети ризиків.
- **Питання для самоконтролю:**
    1. Чому RUP називають процесом, що керується ризиками (Risk-driven)?
    2. Як відрізняється фокус управління ризиками на фазах Inception та Transition?

Тема 4. Інструментальна підтримка та автоматизація етапів RUP

Мета: Ознайомитися з сучасними інструментами (включаючи AI-рішення), що здатні автоматизувати артефакти та робочі процеси (Workflows) RUP.

- **Завдання для самостійного опрацювання:**
    1. Дослідити інструменти для трасування вимог (Requirements Traceability) від ініціації до тестування (напр., Jira, Azure DevOps).
    2. Оцінити можливості використання AI-асистентів для генерації шаблонів UML-діаграм, документації та первинного коду на основі Use Cases.
    3. Налаштувати базове середовище (наприклад, за допомогою Docker-контейнерів) для розгортання архітектурного прототипу.
- **Питання для самоконтролю:**
    1. Які артефакти RUP можна повністю або частково генерувати автоматично?
    2. Як забезпечити контроль версій для моделей UML?

МАТЕРІАЛИ ДЛЯ САМОСТІЙНОГО ВИВЧЕННЯ

Аналітичне моделювання та виділення класів аналізу. Роль та стратегічне значення аналітичного моделювання.

Проектування класів та реалізація діаграми класів. Детальний дизайн інтерфейсу користувача. Побудова повнофункціонального проекту програмної системи.

Аналітичне моделювання має стратегічно важливе значення, оскільки на цьому етапі відбувається спроба змоделювати основну поведінку програмної системи.

- **Коли це відбувається?** Основна аналітична робота починається наприкінці фази **Початок (Inception)**. Поряд із визначенням вимог, аналіз є головною задачею фази **Уточнення (Elaboration)**.
- **Яка мета?** Створення моделей, які відображають потрібну поведінку системи. Ціль робочого потоку аналізу — створення аналітичної моделі, яка фокусується на тому, **ЩО** повинна робити система. Деталі того, **ЯК** вона буде це робити, залишаються для етапу проєктування.
- **Зв'язок із вимогами:** Аналіз суттєво перетинається з визначенням вимог. Зазвичай необхідно провести аналіз вимог, щоб зробити їх зрозумілішими та виявити всі опущення або спотворення.

2. Артефакти аналізу та Метамодель

У робочому потоці аналізу створюються два ключових артефакти:

1. **Класи аналізу:** ключові поняття в бізнес-сфері (предметній області).
2. **Реалізації прецедентів:** ілюструють, як екземпляри класів аналізу можуть взаємодіяти для реалізації поведінки системи, описаної варіантами використання (прецедентами).

Аналітичну модель можна представити у вигляді пакета, який містить один або більше вкладених пакетів аналізу.

Робочий потік аналізу в UP (Unified Process) включає такі діяльності:

- Архітектурний аналіз;
- Аналіз прецеденту;
- Аналіз класу;
- Аналіз пакета.

3. Практичні правила побудови аналітичної моделі

Оскільки всі системи різні, знайти універсальний підхід складно. Проте існують перевірені правила створення якісної аналітичної моделі:

- **Оптимальний розмір:** Аналітична модель системи середнього розміру і складності включає від 50 до 100 класів.
- **Фокус на предметній області:** Включайте ЛИШЕ ті класи, які є частиною словника предметної області. Уникайте додавання проєктних класів (наприклад, підключення до БД), оскільки це артефакти області рішення.
- **Мова бізнесу:** Модель завжди створюється мовою відповідної сфери діяльності. (Наприклад, класи Customer, Order, ShoppingBasket).
- **«Розповідайте історію»:** Кожна діаграма повинна розкривати певні важливі частини поведінки системи.
- **Мінімізація зв'язності (Low Coupling):** Кожна асоціація між класами збільшує їх зв'язність, тому її слід мінімізувати.
- **Природне наслідування:** Використовуйте наслідування лише тоді, коли існує природна ієрархія абстракцій. Ніколи не застосовуйте його просто для повторного використання коду.
- **Зрозумілість для стейкхолдерів:** Модель має бути простою і корисною для всіх зацікавлених сторін.

💡 **Головне правило:** Модель повинна бути простою! Всередині будь-якої складної моделі можна знайти просту, якщо не занурюватися в деталі реалізації завчасно.

4. Виділення класів аналізу (Аналіз прецеденту)

Результатом робочого потоку «Аналіз прецеденту» є класи аналізу та реалізації прецедентів.

Вхідні дані для аналізу прецеденту:

- **Бізнес-модель** (якщо є) — чудове джерело вимог.
- **Модель вимог** — функціональні вимоги підкажуть прецеденти та акторів.
- **Модель прецедентів** — базові сценарії використання.
- **Опис архітектури** — високорівневе бачення системи.

Що таке класи аналізу?

Це класи, які представляють чітку абстракцію предметної області (problem domain) і проєктуються на реальні бізнес-поняття (відповідно до бізнес-правил).

Завдання ОО-аналітика — прояснити безладні прикладні поняття і перетворити їх у те, що може стати основою для класу. Правильне виявлення класів аналізу — це ключ до успішного об'єктно-орієнтованого аналізу та проєктування.

Анатомія класу аналізу

Класи аналізу містять лише ключові атрибути та обов'язки на дуже високому рівні абстракції.

Мінімальна форма класу аналізу включає:

- **Ім'я** (обов'язково).
- **Атрибути** (імена обов'язкові, типи — необов'язкові).
- **Операції** (на рівні аналізу це лише приблизні формулювання обов'язків).
- **Видимість, стереотипи, помічені значення** (необов'язково, лише якщо це покращує розуміння).

5. Ознаки хорошого класу аналізу

Хороший клас аналізу можна охарактеризувати за такими критеріями:

1. **Ім'я відображає призначення** (наприклад, ShoppingBasket — зрозуміло, WebSiteVisitor — занадто абстрактно).
2. **Є чіткою абстракцією**, що моделює один конкретний елемент предметної області.
3. **Має невеликий набір обов'язків.** Обов'язок — це контракт або сервіс, який клас пропонує іншим.
4. **Високе зчеплення (High Cohesion):** Всі обов'язки спрямовані на реалізацію однієї мети (наприклад, кошик повинен додавати/видаляти товари, але НЕ повинен обробляти платежі чи друкувати чеки).
5. **Низька зв'язність (Low Coupling):** Клас взаємодіє лише з необхідним мінімумом інших класів.

6. Практичні правила створення класів (Чек-лист аналітика)

- **3–5 обов'язків на клас.** Прагніть зберегти максимальну простоту.
- **Жоден клас не є ізольованим.** Класи повинні взаємодіяти для досягнення результату.
- **Остерігайтеся множини дрібних класів.** Якщо клас має 1-2 обов'язки, можливо, його слід об'єднати з іншим.
- **Уникайте величезних класів.** Якщо клас має більше 5 обов'язків, спробуйте розбити його на кілька менших (делегування).

- **Остерігайтеся «функтоїдів».** Процедурна функція, яка видається за клас (наприклад, клас, що має лише одну операцію doIt()).
- **Уникайте всемогутніх класів.** Якщо в імені класу є слова System або Controller, і він робить усе підряд — це архітектурна помилка. Розбийте його обов'язки на зв'язкові підмножини.
- **Уникайте «глибоких» дерев наслідування.** В аналізі наслідування застосовується лише там, де є чітка ієрархія бізнес-сутностей. (Більше 3 рівнів наслідування в аналізі — це погано).

7. Методика виявлення класів: Аналіз «Іменник / Дієслово»

Універсального алгоритму для виявлення класів не існує. Проте найдієвішим базовим методом є аналіз тексту технічного завдання або інтерв'ю зі стейкхолдерами.

Суть методу:

- **Іменники та іменні групи** вказують на потенційні **Класи** або їхні **Атрибути**.
- **Дієслова та дієслівні групи** вказують на **Обов'язки (відповідальності)** та **Операції** класу.

На що звернути увагу:

- *Синоніми та омоніми:* можуть стати причиною дублювання або появи хибних класів.
- *Приховані класи:* абстракції, які властиві предметній області, але не згадуються прямо. Наприклад, клієнт каже "бронювати", "скасувати бронь", але ніде не каже слово "Замовлення" (Order). Поява такого класу часто структурує всю систему і вирішує архітектурні проблеми.

Базові концепції уніфікованого процесу
Чотири «П»: Персонал, Проект, Продукт і Процес

Підсумком проекту з розробки ПЗ є продукт, у створенні якого бере участь велика кількість різних людей. Процес розробки спрямовує зусилля персоналу, виступаючи як шаблон із розписаними кроками для виконання проекту. Цей процес автоматизується за допомогою відповідного інструментарію (утиліт).

Основні поняття розробки ПЗ:

Поняття	**Визначення**
Персонал (People)	Архітектори, розробники, тестувальники, менеджери проектів, замовники та користувачі. Це реальні люди (рушійна сила проекту), на відміну від абстрактної концепції «співробітник».
Проект (Project)	Організаційна сутність, за допомогою якої відбувається управління розробкою. Результатом проекту є випущений продукт.
Продукт (Product)	Сукупність артефактів, що створюються протягом життєвого циклу проекту (моделі, тексти програм, виконувані файли, документація).
Процес (Process)	Повний набір видів діяльності, необхідних для перетворення вимог користувача у продукт. Є своєрідним шаблоном для створення проекту.
Утиліти / Інструменти (Tools)	Програмне забезпечення, що використовується для автоматизації визначених у процесі видів діяльності.

Вплив процесу на персонал

Персонал вирішує все: фінансує продукт, планує розробку, створює, тестує та отримує вигоду. Процес розробки має бути орієнтованим на персонал і забезпечувати зручність роботи. Спосіб організації проекту суттєво впливає на команду через наступні аспекти:

- **Здійснимість (реалістичність) проекту:** Ітеративний підхід дозволяє оцінити виконуваність проекту на ранніх стадіях. Роботу над нездійсненними проектами можна припинити вчасно, зменшуючи моральні втрати команди.
- **Управління ризиками:** Дослідження та пом'якшення ризиків на ранніх стадіях підвищує ефективність персоналу, усуваючи відчуття невизначеності.
- **Структура команди:** Максимальна ефективність досягається у невеликих групах (6–8 осіб). Вдала архітектура з чіткими інтерфейсами дозволяє розподілити роботу між такими малими групами.
- **План проекту:** План має бути реалістичним. Нереалістичні плани стрімко знижують продуктивність та бойовий дух команди.

- **Зрозумілість проекту:** Кожен учасник бажає бачити загальну картину. Опис архітектури надає таку можливість усім залученим особам.
- **Відчуття закінченості:** Часті відгуки від користувачів в ітеративному життєвому циклі (ЖЦ) збільшують темп робіт та підсилюють відчуття завершеності проміжних етапів.

Співробітники, обов'язки та ролі

Завдання Уніфікованого процесу (УП) — спрямувати розробників на правильне визначення вимог, вибір надійної архітектури та створення складних систем з найменшими витратами часу та ресурсів.

Трансформація ресурсу в співробітника:

Компанія переводить людину зі статусу абстрактного «ресурсу» на конкретну позицію **«співробітника».**

- **Співробітник (Worker):** Це позиція або роль, яку людина відіграє в розробці ПЗ (наприклад, архітектор, інженер з компонентів, тестувальник інтеграції).
- **Екземпляр співробітника:** Кожен співробітник асоціюється з певним набором видів діяльності. Для ефективної роботи йому потрібна інформація про те, як виконувати ці дії та як його роль співвідноситься з іншими. Співробітник може бути представлений як однією особою, так і групою (наприклад, архітектурний відділ).
- **Обов'язки та навички:** Менеджер проекту повинен співвідносити реальні навички людей з необхідною компетенцією для позиції співробітника. Протягом проекту одна людина може змінювати ролі (наприклад, специфікатор вимог згодом стає інженером з компонентів для цієї ж підсистеми, що мінімізує втрату контексту).

Послідовність самостійного вивчення UML:

1. **Концептуальне моделювання:** Вивчити правила побудови Use Case Diagram. Звернути особливу увагу на різницю між зв'язками <<include>> та <<extend>>.
2. **Статичне моделювання:** Розібрати Class Diagram. Опанувати концепції множинності (Multiplicity) та навчитися відрізняти агрегацію від композиції.
3. **Динамічне моделювання:** Опрацювати Sequence Diagram. Дослідити використання комбінованих фрагментів (alt, opt, loop, par) для моделювання складних алгоритмічних розгалужень.

4. **Моделювання інфраструктури:** Ознайомитися з Component Diagram та Deployment Diagram для підготовки до фінальних етапів проєктування фізичної архітектури.

Рекомендований інструментарій для самостійної практики:

- Окрім стаціонарних середовищ (IBM Rational Software Architect), для самостійних тренувань рекомендується використовувати хмарні інструменти (draw.io, Lucidchart) або текстові генератори діаграм (PlantUML, Mermaid.js), які активно застосовуються в сучасних репозиторіях (GitHub/GitLab).

Життєвий цикл проекту та артефакти

Проект розробки ПЗ є серією змін, що протікають через цикли, фази та ітерації.

- **Серії ітерацій:** Кожна ітерація є міні-проектом, в ході якого реалізується набір взаємопов'язаних варіантів використання (ВВ) або зменшуються певні ризики. Виконуються всі базові робочі процеси: збір вимог, проектування, реалізація, тестування.
- **Організаційний шаблон:** Процес створює зразок роботи, визначаючи необхідні типи співробітників та артефакти, з якими вони працюють.

Поняття Артефакту:

Артефакт — це загальна назва для будь-якої інформації, що створюється, змінюється або використовується розробниками (моделі UML, вихідний код, тексти, прототипи).

1. **Технічні артефакти:** Створюються на фазах аналізу, проектування, розробки та тестування.
2. **Артефакти управління:** Бізнес-плани, плани розробки та ітерацій, плани розподілу співробітників тощо.

Система як набір моделей

Модель — це абстракція, що описує систему з певної точки зору і на певному рівні деталізації. Побудова системи зводиться до побудови моделей.

- **Модель Варіантів Використання (ВВ):** *Зовнішнє представлення* системи. Описує, що система робить для користувачів (актантів). Містить варіанти використання.

• **Модель Проектування:** *Внутрішнє представлення* системи. Описує, як система побудована. Містить підсистеми, інтерфейси, класи та кооперації.

Зв'язки трасування (Traceability):

Система — це не лише набір моделей, а й зв'язки між ними. Зв'язок трасування пов'язує елементи однієї моделі з елементами іншої (наприклад, ВВ із відповідним елементом моделі аналізу або класи з компонентами). Це полегшує перенесення змін між моделями та забезпечує цілісність.

Робочі процеси (Workflows) та їх спеціалізація

Робочі процеси формуються за такою схемою:

1. Визначення типів співробітників.
2. Визначення артефактів, які вони створюють.
3. Опис потоку робіт (як співробітники взаємодіють через артефакти).

Приклад: У робочому процесі «Визначення вимог» беруть участь системний аналітик, архітектор, специфікатор ВВ. Вони створюють модель ВВ. В термінах UML такий процес розглядається як стереотип кооперації.

Спеціалізація процесу:

УП є узагальненим каркасом, який кожна команда повинна адаптувати (спеціалізувати) під власні потреби. Фактори спеціалізації:

• **Організаційні:** Культура команди, досвід.
• **Предметної області:** Специфіка бізнес-процесів замовника.
• **Життєвого циклу:** Час виходу на ринок (Time-to-market).
• **Технічні:** Мови програмування, бази даних, існуючі каркаси (frameworks).

Інструментальна підтримка (Tools)

Засоби підтримки та процес є нерозривним цілим: процес керує інструментами, а інструменти спрямовують розвиток процесу.

Базові категорії засобів автоматизації:

• **Засоби управління вимогами:** Зберігання, відстеження статусів та трасування вимог до тестових випадків.

- **Засоби візуального моделювання:** Автоматизація роботи з UML. Забезпечують постійну узгодженість моделі та коду (синтаксичні та семантичні перевірки).
- **Засоби програмування:** Редактори, компілятори, відлагоджувачі.
- **Засоби контролю якості:** Автоматизація тестування (регресійного, навантажувального, GUI-тестування).
- **Допоміжні засоби:** Системи контролю версій (VCS), управління конфігураціями, системи відстеження дефектів (Bug trackers).

Архітектура програмного забезпечення

Архітектура — це основа системи, що концентрується на її структурних елементах (підсистемах, класах, вузлах) та їх кооперації через інтерфейси.

- **Формування:** Архітектура розробляється ітеративно на основі найбільш значущих (архітектурно-важливих) варіантів використання та нефункціональних вимог.
- **Опис архітектури:** Це вибірка з різних моделей системи (моделі ВВ, аналізу, проектування, розгортання), яка містить інформацію, критично важливу для розуміння системи всіма учасниками проекту.

Для поглибленого вивчення дисципліни студентам пропонується обрати одну з тем для самостійного дослідження. Результати оформлюються у вигляді аналітичного звіту (реферату) або презентації для виступу на практичному занятті.

Блок 1. Еволюція методологій та процесів

1. Адаптація класичного RUP до сучасних реалій: Agile RUP (AUP) та OpenUP.
2. Порівняльний аналіз процесів управління вимогами: Use Cases (RUP) проти User Stories (Scrum/Agile).
3. Масштабування гнучких методологій для великих корпоративних систем (SAFe, LeSS, Nexus).
4. Роль бізнес-аналітика та архітектора в різних методологіях розробки ПЗ.

Блок 2. Архітектура та проєктування 5. Предметно-орієнтоване проєктування (Domain-Driven Design, DDD) як логічне продовження об'єктно-орієнтованого аналізу. 6. Мікросервісна архітектура: переваги, недоліки та межі застосування. 7. Безсерверні обчислення (Serverless Architecture) та їх вплив на проєктування систем. 8. Модель C4 як сучасна альтернатива та доповнення до

класичних діаграм UML: підхід «Architecture as Code». 9. Патерни проєктування (GoF) та їх реалізація у сучасних веб-фреймворках.

Блок 3. Інженерні практики: DevOps та AI 10. Інтеграція практик DevOps (CI/CD) у життєвий цикл розробки програмного забезпечення. 11. Технології контейнеризації (Docker) та оркестрації (Kubernetes) на етапі розгортання системи. 12. Впровадження інструментів генеративного штучного інтелекту (AI Engineering) для автоматизації рефакторингу та написання коду на фазі «Побудова». 13. DevSecOps: забезпечення безпеки програмного забезпечення на етапі проєктування архітектури.

Оскільки базовим інструментом для виконання лабораторних робіт є уніфікована мова моделювання (UML), студентам необхідно самостійно опрацювати специфікацію базових діаграм.

ПИТАННЯ ДЛЯ САМОКОНТРОЛЮ

Ці питання охоплюють ключові аспекти дисципліни та призначені для самоперевірки знань при підготовці до заліку або екзамену.

Основи інженерії ПЗ та RUP:

1. Що таке життєвий цикл програмного забезпечення? Назвіть його основні моделі.
2. Які три основні характеристики (принципи) визначають методологію RUP?
3. Охарактеризуйте мету та основні артефакти фази «Початковий етап» (Inception).
4. У чому полягає різниця між фазою «Уточнення» (Elaboration) та «Побудова» (Construction)?
5. Як RUP підходить до управління ризиками проєкту?

Аналіз та Управління вимогами:

6. Дайте визначення функціональним та нефункціональним вимогам. Наведіть приклади.

7. Що таке модель FURPS+? Розшифруйте кожну літеру абревіатури.

8. Які складові повинна містити якісна текстова специфікація варіанта використання (Use Case Specification)?

9. Для чого використовується Матриця трасування вимог (Requirements Traceability Matrix)?

Проєктування та Архітектура (UML/SOLID/GRASP):

10. У чому полягає різниця між об'єктом і класом в об'єктно-орієнтованому підході? 11. Поясніть сутність патерну BCE (Boundary-Control-Entity).

12. Перелічіть п'ять принципів SOLID та коротко поясніть кожен із них.

13. У чому суть патернів Information Expert та Creator згідно з GRASP?

14. З яких п'яти представлень складається архітектурна модель «4+1» Філіпа Крачтена?

15. Які критерії визначають «хорошу» архітектуру програмної системи (зв'язність, зчеплення, масштабованість)?

Реалізація, Тестування та Впровадження:

16. У чому полягає різниця між верифікацією та валідацією програмного забезпечення?

17. Поясніть концепцію безперервної інтеграції (Continuous Integration).

18. Для чого потрібна діаграма розгортання (Deployment Diagram) та які її основні елементи?

19. Як технології контейнеризації (наприклад, Docker) допомагають уникнути проблеми "На моєму комп'ютері це працює"?

РЕКОМЕНДОВАНІ ІНСТРУМЕНТАРІЇ ТА ДОДАТКОВІ РЕСУРСИ

Програмне забезпечення для візуального моделювання та проєктування (CASE-засоби):

- IBM Rational Software Architect Designer — класичний комплексний інструмент для об'єктно-орієнтованого аналізу, проєктування та генерації коду в екосистемі RUP.
- PlantUML або Mermaid.js — інструменти для створення UML-діаграм на основі тексту (підхід «Архітектура як код» / Architecture as Code).
- Structurizr — засіб для багаторівневого моделювання архітектури програмного забезпечення за моделлю C4.
- Draw.io (diagrams.net) або Lucidchart — гнучкі хмарні сервіси для створення концептуальних UML-діаграм, схем баз даних та діаграм розгортання.

Середовища розробки (IDE) для реалізації архітектурних рішень:

- Локальні IDE: Visual Studio Code, а також спеціалізовані інтегровані середовища розробки від JetBrains (WebStorm, PyCharm, IntelliJ IDEA).
- Хмарні середовища розробки (Cloud IDE): Project IDX або GitHub Codespaces — для швидкого прототипування, ізольованого тестування та зручної командної роботи над кодом.

Платформи та фреймворки (для практичної реалізації на фазі «Побудова»):

- Серверна частина (Backend): платформи Node.js та Python для реалізації серверної бізнес-логіки, проєктування API та мікросервісної архітектури.
- Клієнтська частина (Frontend): бібліотека React для проєктування користувацьких інтерфейсів та реалізації архітектурних патернів (наприклад, MVC, Flux/Redux).
- Хмарна інфраструктура (BaaS / Serverless): платформи на зразок Firebase для швидкого розгортання баз даних, налаштування автентифікації та побудови безсерверної (serverless) архітектури.

Інструменти DevOps та контейнеризації (для фаз Побудови та Впровадження):

- Docker: для контейнеризації, ізоляції середовищ та пакування мікросервісів.
- Git та платформи GitHub / GitLab: для управління версіями вихідного коду (Version Control), командної взаємодії та налаштування базових конвеєрів безперервної інтеграції і доставки (CI/CD пайплайнів).

Інструменти на базі штучного інтелекту (AI-асистенти):

- Генеративний ШІ: використання сучасних AI-інструментів для допомоги в аналізі архітектурних патернів, рефакторингу, оптимізації та генерації шаблонного (boilerplate) коду. *Примітка: використання таких інструментів має суворо відповідати принципам академічної доброчесності.*

Сучасні методології та архітектурні підходи (як еволюція та доповнення RUP):

- Інтеграція Agile-практик (Scrum, Kanban): адаптація ітеративної природи RUP до сучасних гнучких методологій розробки для швидшої доставки цінності.
- Domain-Driven Design (DDD): предметно-орієнтоване проєктування для глибокого аналізу бізнес-вимог, визначення обмежених контекстів (Bounded Contexts) та проєктування меж мікросервісів.
- Cloud-Native підходи: проєктування сучасних розподілених систем, мікросервісних (Microservices) та безсерверних (Serverless) архітектур.
- DevOps / DevSecOps: впровадження інженерних практик безперервної інтеграції, тестування та розгортання, а також підходу «Інфраструктура як код» (IaC).
- Модель C4: використання багаторівневого підходу (Context, Containers, Components, Code) як сучасної альтернативи або доповнення до класичних діаграм UML для документування архітектури програмного забезпечення.

Обчислювальні ресурси:

- Стандартний персональний комп’ютер або ноутбук із характеристиками, достатніми для запуску сучасних середовищ розробки (IDE) та систем віртуалізації (наприклад, Docker Desktop). Обов’язковою умовою є наявність стабільного доступу до мережі Інтернет для роботи з хмарними сервісами, віддаленими репозиторіями та системами управління проєктами.

ВИСНОВКИ

Протягом перших років роботи в IT-індустрії розробники зазвичай фокусуються на написанні програмного коду та реалізації локальних рішень для конкретних задач. Однак із поглибленням професійного досвіду стає очевидним, що більшість критичних помилок і вузьких місць набагато ефективніше усувати ще на етапі аналізу та проєктування програмного забезпечення, за допомогою вдалого архітектурного рішення.

Патерни та архітектурні стилі — це перевірені часом типові плани, які забезпечують елегантне вирішення проблем, з якими ви будете стикатися знову і знову у своїй кар’єрі інженера. Проте успішне застосування патернів неможливе без системного підходу до розробки, який і забезпечує методологія ітеративного проєктування.

Протягом виконання лабораторного практикуму з дисципліни ви пройшли весь шлях створення програмного продукту — від початкового етапу (Inception) до фази впровадження (Transition). Ви навчилися:

- проводити глибокий аналіз предметної області, ідентифікувати стейкхолдерів та грамотно управляти функціональними й нефункціональними вимогами;
- будувати візуальні моделі за допомогою мови UML для документування концептуальних варіантів використання, структури класів та динаміки взаємодії об'єктів;
- правильно розподіляти обов'язки між об'єктами (застосовуючи принципи SOLID та патерни GRASP) для забезпечення низької зв'язності та високого зчеплення архітектури;
- використовувати сучасні CASE-засоби та середовища візуального моделювання (наприклад, IBM Rational Software Architect та аналоги) для проєктування та автоматизованої генерації базового каркаса коду;
- проєктувати фізичну архітектуру та готувати систему до розгортання, розуміючи роль контейнеризації та сучасних DevOps-практик;
- ідентифікувати технічні ризики, будувати матриці трасування та планувати стратегію приймального тестування для валідації кінцевого продукту.

Комплексне бачення життєвого циклу розробки та набуті практичні навички об'єктно-орієнтованого проєктування є міцним фундаментом для подальшого професійного зростання. Вони дозволять вам еволюціонувати від позиції рядового розробника (програміста) до рівня Software Architect, здатного проєктувати гнучкі, масштабовані та надійні корпоративні системи, що приносять реальну цінність бізнесу.

РЕКОМЕНДОВАНІ ДЖЕРЕЛА

Базова література:

1. Kruchten, P. (2003). *The Rational Unified Process: An Introduction* (3rd ed.). Addison-Wesley Professional.
2. Sommerville, I. (2020). *Software Engineering* (11th ed.). Pearson.
3. Pressman, R. S., & Maxim, B. R. (2019). *Software Engineering: A Practitioner's Approach* (9th ed.). McGraw-Hill Education.
4. Wiegers, K., & Beatty, J. (2023). *Software Requirements* (4th ed.). Microsoft Press.
5. Larman, C. (2004). *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development* (3rd ed.). Prentice Hall.

Допоміжна література:

6. Martin, R. C. (2017). *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Prentice Hall.
7. Richards, M., & Ford, N. (2020). *Fundamentals of Software Architecture: An Engineering Approach*. O'Reilly Media.
8. Newman, S. (2021). *Building Microservices: Designing Fine-Grained Systems* (2nd ed.). O'Reilly Media.
9. Brown, S. (2020). *Software Architecture for Developers: Visualise, document and explore your software architecture*. Leanpub.
10. Vernon, V. (2016). *Domain-Driven Design Distilled*. Addison-Wesley Professional.
11. Rubin, K. S. (2022). *Essential Scrum: A Practical Guide to the Most Popular Agile Process*. Addison-Wesley.
12. Kim, G., Humble, J., Debois, P., Willis, J., & Forsgren, N. (2021). *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations* (2nd ed.). IT Revolution Press.
13. Farley, D. (2021). *Modern Software Engineering: Doing What Works to Build Better Software Faster*. Addison-Wesley.

Інформаційні ресурси в мережі Інтернет:

14. Офіційна специфікація UML (Unified Modeling Language) версії 2.5.1 від OMG (Object Management Group). [Електронний ресурс] — Режим доступу: https://www.omg.org/spec/UML/2.5.1/PDF
15. Офіційний портал методології C4 (The C4 model for visualising software architecture). [Електронний ресурс] — Режим доступу: https://c4model.com/
16. ISO/IEC/IEEE 29148:2018. Systems and software engineering — Life cycle processes — Requirements engineering. [Електронний ресурс] — Режим доступу: https://www.iso.org/standard/72089.html
17. Офіційна документація та керівництва з використання сучасних екосистем розробки (наприклад, React, Node.js), систем управління базами даних та хмарних платформ (залежно від обраного технологічного стека у проєкті).