Шишацька Олена Володимирівна, к.ф.-м.н., доцент

ІНТЕГРАЦІЯ ТЕОРЕТИЧНОГО ТА ПАРАДИГМАЛЬНОГО ПІДХОДІВ У ПІДГОТОВЦІ БАКАЛАВРІВ ІНФОРМАТИКИ

У статті висвітлено досвід поєднання теоретичного та парадигмального підходів у підготовці бакалаврів комп'ютерних наук (ОПП «Інформатика») в КНУ імені Тараса Шевченка. На прикладі вибіркової дисципліни «Парадигми та технології програмування» продемонстровано методичні можливості практичної реалізації теоретичного апарату обов'язкового курсу «Теорія програмування» (формалізація мов, алгебра програм, коректність) у функціональному, логічному й об'єктному стилях програмування. Проаналізовано зміст курсів, зв'язок між ними, форми інтеграції та результати навчання. Наведено таблиці відповідності понять і зіставлення результатів обох дисциплін.

Ключові слова: теорія програмування, парадигми програмування, інтеграція підходів, бакалавр інформатики, компетентності, результати навчання, формалізація програм, коректність програм.

The article highlights the experience of combining theoretical and paradigmatic approaches in training Bachelor students of Computer Science (EPP "Informatics") at Taras Shevchenko National University of Kyiv. Using the elective course 'Programming Paradigms and Technologies' as a case study, the paper explores methodological approaches to putting the theoretical framework of the mandatory 'Theory of Programming' course (language formalization, program algebra, correctness) into practice within functional, logic, and object-oriented programming paradigms. The paper analyzes course contents, the connection between them, forms of integration, and learning outcomes, providing tables of conceptual correspondence and a comparison of the results of both disciplines.

Keywords: theory of programming, programming paradigms, integration of approaches, Bachelor of Informatics, competences, learning outcomes, program formalization, program correctness.

Вступ

Підготовка фахівців у галузі інформаційних технологій – одна з найдинамічніших і водночас найбільш суперечливих сфер сучасної університетської освіти. З одного боку, практикоорієнтованість і швидка зміна технологічного ландшафту вимагають від випускника здатності

негайно вирішувати прикладні задачі. З іншого – фундаментальна природа програмування як наукової дисципліни вимагає від фахівця глибокого розуміння теоретичних основ, без яких неможлива ані обґрунтована архітектура складних систем, ані критична оцінка нових обчислювальних парадигм.

Ця суперечність добре відома у світовій педагогіці комп'ютерних наук. Ще наприкінці XX ст. Е. Дейкстра та Ч. Хоар наголошували, що програмування є математичною дисципліною і що без відповідної теоретичної культури фахівець приречений на помилки, яких принципово не може виявити інтуїтивне мислення [5]. Водночас ізольоване викладання теорії без чіткого зв'язку з практикою перетворює формальні курси на незрозумілі та демотивуючі для студентів.

У сучасному контексті ця проблема набуває нового виміру: поряд із класичними парадигмами програмування – об'єктно-орієнтованою, функціональною, логічною – активно розвиваються реактивне, агентне, квантове програмування. Фахівець в галузі комп’ютерних наук, що не отримав системного уявлення про природу парадигм і їхній теоретичний фундамент, виявляється беззахисним перед цим різноманіттям: він може «вивчити» чергову мову, але не здатний обрати адекватну парадигму для задачі і обґрунтувати свій вибір.

В освітньо-професійній програмі «Інформатика» обов'язковий курс «Теорія програмування» (ТП) забезпечує фундаментальну математично-методологічну культуру мислення, яка знаходить пряме прикладне розгортання в дисциплінах спеціалізації. Зокрема, у вибірковому блоці кафедри теорії та технології програмування дисципліна «Парадигми та технології програмування» (ПТП) слугує показовим майданчиком для систематичної матеріалізації категоріального апарату теорії.

Мета статті: висвітлення досвіду та обґрунтування методичної інтеграції теоретичного та парадигмального підходів у підготовці бакалаврів інформатики. Завдання статті: охарактеризувати сутність і змістове наповнення кожного з підходів; виявити структурно-логічні зв'язки між дисциплінами на рівні конкретних понять і конструкцій; описати методичну реалізацію інтеграції в навчальному процесі; проаналізувати результати навчання та компетентності, що формуються на перетині двох підходів.

Теоретичний підхід у підготовці бакалаврів: сутність і роль

Дисципліна «Теорія програмування» як обов'язкова складова освітньо-професійної програми «Інформатика» викладається у 5-му семестрі третього курсу бакалаврату в обсязі 4 кредитів ЄКТС (120 годин): 28 годин лекцій, 28

годин практичних занять та 62 години самостійної роботи [1]. Курс структурований у дві змістовні частини: перша охоплює формалізацію мови програмування та основні програмні поняття, друга – синтаксис і семантику програм, рекурсивні програми та огляд формальних підходів до проєктування програмних систем.

Концептуальна основа курсу – авторський навчальний посібник [2], методологічна база якого полягає в наступному твердженні: суть теорії програмування – це категоріальний (поняттєвий) аналіз процесу програмування. Поняття визначаються у єдності двох моментів: змісту (інтенсіоналу) та обсягу (екстенсіоналу). Тому для теорії програмування найважливішим є не конкретна програма, а відношення між програмними поняттями – логічна, інтенсіональна складова [2].

Теоретичний підхід ґрунтується на трьох фундаментальних принципах. По-перше, принцип єдності теорії та практики: розробка теорії ґрунтується на науковому дослідженні та аналізі практики, після чого наукові теорії втілюються в практику через відповідні технології. Цей цикл – практика → наукове дослідження → теорія → технологія → практика – не лише наукова схема, а й дидактичний орієнтир [2]. По-друге, принцип сутнісного підходу: на відміну від феноменологічного підходу, що описує зовнішні прояви процесів, сутнісний підхід спрямований на розкриття їх внутрішніх причин і зв'язків. По-третє, принцип від абстрактного до конкретного: побудова теорії здійснюється від простих абстракцій до складних програмних систем, що відповідає основним методам побудови програм – систематичному, зверху-вниз, структурному [2].

Практична реалізація цих принципів у курсі починається з формалізації простої мови програмування SIPL (Simple Imperative Programming Language). Мова SIPL – варіант класичних мов структурного програмування та охоплює присвоєння, послідовне виконання, розгалуження й цикл. Для SIPL у курсі будується повна формалізація:

1) синтаксис мови задається за допомогою форми Бекуса–Наура (БНФ); розглядаються дерева синтаксичного виводу, питання однозначності граматик, пріоритет операцій;

2) семантика мови задається через алгебру даних і функціональні алгебри; операціями алгебр є композиції, що формалізують засоби конструювання програм: суперпозиція (Sn), присвоєння (ASx), послідовне виконання (•), умовний оператор (IF), цикл (WH);

3) для кожної програми будується семантичний терм – її образ в алгебрі програм A_Prog;

4) на основі побудованих формалізацій доводиться часткова та повна коректність програм.

Ключовий результат цього аналізу – усвідомлення студентом того, що «без наявності формальної семантики та синтаксису ми не могли б навіть говорити про доведення різних властивостей програм та, зокрема, їх коректності. На інтуїтивному рівні розуміння програм може бути лише інтуїтивне обґрунтування властивостей програм» [2]. Це твердження методологічно принципове: воно переводить поняття коректності з рівня інтуїції на рівень строгої математичної теорії.

Центральний математичний результат другої частини курсу: теорема про найменшу нерухому точку – фундаментальний результат, що лежить в основі семантики рекурсивних програм. Студент опановує поняття повних впорядкованих множин, монотонних і неперервних операторів, ієрархію граматик Хомського, автоматні формалізми сприйняття мов, контекстно-вільні граматики та методи семантичного аналізу – аксіоматичну, денотаційну, композиційну семантики [1].

Схема формалізації мови SIPL узагальнюється до загальної схеми: мова L задається як трійка (Synt, Sem, interpretation), де Synt – опис синтаксичного аспекту, Sem – опис семантичного аспекту (смислу програм), а interpretation: Synt → Sem – інтерпретація, що кожній програмі зіставляє її смисл. Саме ця схема, оволодівши якою студент дійсно розуміє, що таке мова програмування, стає містком до глибокого розуміння будь-якої парадигми.

Важлива педагогічна особливість курсу – форма проведення практичних занять. Зважаючи на складність матеріалу та необхідність формування навичок математичного мислення, практичні заняття передбачають обговорення розв'язань задач з залученням студентів до дискусії. У зв'язку з цим практичні заняття доцільно проводити з академічною підгрупою чисельністю до 15 осіб. Такий формат дає змогу реалізувати сократівський метод: задачі не лише розв'язуються, а й обговорюються, аналізуються з різних боків, обґрунтовуються.

Базою для опанування ТП є попередні математичні дисципліни циклу загальної та професійної підготовки («Дискретна математика», «Математична логіка», «Теорія алгоритмів»). Своєю чергою, категоріальний апарат курсу ТП виконує інтегративну роль на завершальному етапі бакалаврату: він забезпечує формально-семантичну основу для виконання курсових та кваліфікаційних робіт, дисциплін фахового вибору комп'ютерного спрямування (зокрема парадигмальних та архітектурних спецкурсів), а також створює наукове підґрунтя для дисциплін другого

(магістерського) рівня вищої освіти, пов'язаних із формальними методами верифікації та прикладною логікою [2].

Парадигмальний підхід: багатовимірне мислення програміста

Поняття «парадигма програмування» введено Робертом Флойдом у 1978 р. та розвинуто Томасом Куном у контексті наукових революцій. У програмуванні парадигма – спосіб концептуалізації обчислень, що визначає, як програміст описує задачу, які абстракції він використовує і як вони відображаються на обчислювальний процес. Різні парадигми не конкуруючі – вони є взаємодоповнювальними, кожна з яких найкраще підходить для певного класу задач.

Дисципліна «Парадигми та технології програмування» – вибіркова в рамках блоку «Теорія та технологія програмування» і викладається у 6-му семестрі третього курсу в обсязі 4 кредитів ЄКТС (120 годин): 28 годин лекцій, 14 годин лабораторних робіт та 76 годин самостійної роботи [3]. Курс завершується екзаменом і передбачає три змістові частини та три контрольні роботи у форматі тестів.

Мета курсу – засвоєння базових знань щодо використання різних парадигм програмування: функціональної, логічної, декларативної, об'єктно-орієнтованої; оволодіння навичками проєктування та розробки програмних систем із застосуванням різноманітних парадигм з відповідними структурами даних, механізмами управління та технологіями [3].

Частина 1. Функціональна парадигма (Haskell)

Теоретичною базою функціонального програмування є лямбда-числення Алонзо Черча (1936). Функціональне програмування – стиль, у якому програма є набором математичних функцій без побічних ефектів. Мова Haskell, обрана для вивчення цієї парадигми, є «чистою» функціональною мовою зі строгою статичною типізацією та лінивими обчисленнями. У курсі розглядаються: типізація в Haskell, визначення функцій та механізм карінгу (часткового застосування); визначення функцій кількома рівняннями та зіставлення зі зразком (pattern matching); рекурсивні функції та теорема про нерухому точку; поліморфні типи та параметричний поліморфізм; функції вищих порядків, лямбда-функції, функції curry та uncurry; ліниві обчислення, нескінченні списки та list comprehension; монадні обчислення, вводу-виводу та обробка виключень (монада Maybe) [3]. Ключова дидактична ідея: студент уперше бачить програму не як послідовність команд, що змінюють стан машини, а як систему математичних рівнянь. Це радикально змінює спосіб мислення і готує ґрунт для більш глибокого розуміння семантики, що вивчалася в ТП.

Частина 2. Логічна парадигма (Prolog)

Логічне програмування ґрунтується на формальній логіці предикатів першого порядку: програма – набір аксіом і правил, а виконання – пошук доведення цільового твердження. Мова Prolog, реалізована у варіанті SWI-Prolog, є найпоширенішою мовою логічного програмування.

У курсі вивчаються: синтаксис «чистого» Прологу – факти, правила та запити; декларативна (логічна) та операційна семантики Prolog-програм; метод резолюцій, алгоритм уніфікації, стратегія пошуку та дерево виводу; особливості Prolog-процесорів – механізм відкатів (backtracking) та керування ним; техніки логічного програмування – робота зі списками, моделювання циклів через рекурсію, стеки та черги; логіка в експертних системах – пряма та обернена стратегії пошуку відповіді, подання знань у вигляді правил; експертні системи з поясненнями – відповіді на питання «Як?» та «Навіщо?», робота в умовах неповної визначеності [3].

Методично ця частина найбільш безпосередньо пов'язана з теорією програмування: алгоритм уніфікації, що є операційним ядром Prolog-процесора, детально розглянутий у курсі математичної логіки та зустрічається в ТП при вивченні резолютивних методів. Студент природньо переходить від абстрактного формалізму до живої мови.

Частина 3. Декларативні засоби в ООП (Java / .NET)

Третя частина курсу поширює принципи декларативності на найбільш поширену об'єктно-орієнтовану парадигму. Студенти вивчають: метадані та анотації – рефлексія об'єктів, декларативний стиль у Java, створення та використання власних анотацій; декларативні фреймворки на основі анотацій для IoC (Inversion of Control) та DI (Dependency Injection); сервісно-орієнтована архітектура (SOA) – принципи побудови розподілених систем та еволюція стандартів від SOAP до REST; декларативний опис мережевих інтерфейсів – порівняння контрактного (WSDL) та ресурсного (OpenAPI/Swagger) підходів [3].

Ця частина – приклад того, як декларативні ідеї функціонального та логічного програмування проникають у мейнстрімне ООП. Студент бачить, що «парадигмальна чистота» не є вимогою реального програмування – важливіше розуміти, який саме декларативне виражальний засіб обирається і чому.

Попередні вимоги до ПТП прямо передбачають компетентності, здобуті у ТП та суміжних курсах: знання загальних понять мов програмування (синтаксис, семантика), основ теорії алгоритмів (рекурсивні функції та обчислення), математичної логіки (резольвента, резолютивний

вивід), а також уміння описувати на формальних мовах 1-го порядку твердження щодо предметних областей [3].

Структурно-логічний зв'язок між підходами

Послідовне розміщення дисциплін – ТП у 5-му семестрі, ПТП у 6-му – методично вмотивована архітектура навчального процесу. Теоретичний курс виступає концептуальним фундаментом, на якому парадигмальний курс зводить конкретні «будівлі» обчислювальних стилів.

Ключовим методологічним принципом, що об'єднує обидва курси, є ідея композиційного програмування – підходу, запропонованого В. Н. Редьком і розвиненого у Київській науковій школі [2]. Програма розуміється не як набір інструкцій для машини, а як алгебраїчний вираз у певній програмній алгебрі, де базові функції і правила їх композиції повністю визначають семантику мови. Цей погляд універсальний: і функціональна програма на Haskell, і логічна програма на Prolog, і анотований Java-клас – різні реалізації одного й того ж принципу – декларативного опису обчислень через систему взаємодіючих виразів. Зв'язок між двома підходами виявляється на рівні конкретних понять і конструкцій. Розглянемо ці зв'язки детально.

Теорема про нерухому точку, що є центральним результатом теорії рекурсивних програм, безпосередньо пояснює семантику рекурсивних функцій у Haskell. Функція fix :: (a → a) → a – програмне втілення конструктивного методу знаходження найменшої нерухомої точки; ліниві обчислення Haskell забезпечують коректність цього методу навіть для нескінченних структур даних.

Часткова та повна коректність, що доводяться в ТП для програм мови SIPL, знаходить відображення у системі типів Haskell та монадному програмуванні. Монада Maybe – програмний аналог часткових функцій – явно виражає можливість невизначеності результату – саме те, що в ТП записується через позначку fs(st)↑ (функція не визначена на стані st). Монада Either узагальнює це до обробки помилок.

Найбільш прямий зв'язок між курсами виявляється у логічній парадигмі. Алгоритм уніфікації – операційне ядром Prolog-процесора – конкретизує абстрактне поняття «розв'язання системи рівнянь у термовій алгебрі», з яким студенти ознайомлюються у ТП через формалізм алгебр. Механізм відкатів (backtracking) – реалізація стратегії пошуку в дереві виводу – поняття, безпосередньо пов'язане з методом резолюцій. Факти і правила Prolog – це аксіоми і правила виводу, що вивчалися у математичній логіці та застосовувалися у ТП.

Предметні області і мови специфікацій, що розглядаються у другій частині ТП, набувають конкретного програмного втілення в логіці експертних систем: база знань Prolog є формальним описом предметної області мовою логіки предикатів першого порядку. Механізм пояснень (відповіді на «Як?» та «Навіщо?») реалізує аксіоматичний підхід до верифікації висновків.

Поняття алгебри програм і рівнів абстракції програмних систем, що розглядаються у ТП, практично втілено у третій частині ПТП. Анотації Java і атрибути .NET – декларативний опис метарівня програмної системи; принцип IoC реалізує інверсію залежностей, що є архітектурним аналогом переходу від конкретної до абстрактної алгебри; OpenAPI-специфікація – формальний контракт у дусі мов специфікацій, що вивчалися в ТП (таблиця 1).

Таблиця 1. Відповідність понять теоретичного та парадигмального курсів

Поняття / курс ТП	**Парадигма / курс ПТП**	**Конкретна конструкція або концепція**
Теорема про найменшу нерухому точку	Функціональна (Haskell)	Рекурсивні функції, fix-комбінатор, ліниві обчислення
Часткові функції: fs(st)↑ / fs(st)↓	Функціональна (Haskell)	Монади Maybe та Either; обробка виключень
Поліморфні функції та алгебри	Функціональна (Haskell)	Параметричний поліморфізм, класи типів (type classes)
Уніфікація, резолютивний вивід	Логічна (Prolog)	Механізм зіставлення (matching), запити до БЗ
Стратегія пошуку, дерево виводу	Логічна (Prolog)	Backtracking, керування відкатами (cut)
Предметні області, мови специфікацій	Логічна (Prolog)	Бази знань, ЕС, механізм пояснень («Як?»/«Навіщо?»)
Алгебра програм, метарівень систем	Декларативна в ООП	Анотації, рефлексія, IoC/DI-фреймворки
Формальний контракт / мови специфікацій	Декларативна в ООП	OpenAPI/Swagger, WSDL-специфікації
Рівні абстракції програмних систем	Усі три парадигми	Вибір парадигми залежно від рівня та класу задач

Наведена таблиця не вичерпний перелік, а ілюстрація загального принципу: кожне поняття, введене в ТП як абстрактний теоретичний об'єкт, знаходить конкретне програмне втілення в одній або кількох парадигмах, що

вивчаються в ПТП. Це забезпечує двоспрямований навчальний ефект: теоретичний курс готує студента до парадигмального, а парадигмальний – поглиблює і конкретизує засвоєне в теоретичному. Принципово важливо, що цей зв'язок не штучно сконструйований, а органічно випливає зі спільної математичної природи обох дисциплін: обидві спираються на формальну логіку, теорію множин і алгебраїчні структури як свій фундамент. Студент, що добре засвоїв ТП, приходить на курс ПТП із готовим понятійним апаратом – і не просто «вчить нові мови», а впізнає в нових конструкціях уже відомі теоретичні об'єкти.

Методична реалізація інтеграції в навчальному процесі

Інтеграція теоретичного та парадигмального підходів реалізується через систему взаємопов'язаних методичних механізмів, що охоплюють структуру навчального плану, форми та методи навчання, організацію самостійної роботи та систему оцінювання.

Наступність і спільна математична база

Обидва курси спираються на спільний математичний фундамент – дискретну математику, математичну логіку та теорію алгоритмів. Ця спільна основа унеможливлює сприйняття теоретичного і парадигмального курсів як розрізнених. Студент, що навчився формалізувати предметну область мовами логіки першого порядку та встановлювати істинність формул, ніколи не буде розглядати факти і правила Prolog як «щось нове» – він впізнає в них добре знайомі синтаксичні та семантичні структури.

Вхідні вимоги до ПТП прямо передбачають знання і вміння, сформовані у ТП: загальні поняття мов програмування (синтаксис, семантика, об'єктно-орієнтований стиль), основи теорії алгоритмів (рекурсивні функції та рекурсивні обчислення), основні поняття математичної логіки (зокрема поняття резольвенти та резолютивного виводу), вміння описувати на формальних мовах 1-го порядку твердження щодо предметних областей та властивостей програм [3].

Форми навчання і методи викладання

Обидва курси поєднують лекційну та практичну (лабораторну) складові, проте з різним акцентом і методичним наповненням.

У курсі ТП лекції присвячені систематичному викладенню теоретичного апарату – від формалізації SIPL до семантики рекурсивних програм і сучасних методів розробки. Практичні заняття мають принципово діалогічний характер: студенти не просто відтворюють зразки, а розв'язують задачі формалізації, будують семантичні терми, доводять коректність

програм – і захищають свої рішення перед аудиторією. Проведення практик з половиною групи є необхідною умовою такого діалогу [1].

Проєктне завдання – інтегративний компонент курсу ТП: воно об'єднує знання з обох частин курсу (формалізація і семантика) і вимагає від студента самостійного дослідницького мислення. Студент може замінити проєктне завдання на здачу відповідного сертифіката з курсу, попередньо узгодженого з викладачем [1].

У курсі ПТП лабораторні роботи організовані за принципом наростаючої складності: спочатку студент освоює базові конструкції мови (типи, функції, рекурсія у Haskell; факти та запити в Prolog; анотації в Java), потім переходить до повноцінних програмних рішень – реалізація алгоритмів на списках, побудова міні-ЕС, розробка власних декларативних фреймворків. Захист лабораторних робіт включає усне обговорення прийнятих рішень, що безпосередньо корелює з теоретичним умінням обґрунтовувати властивості програм [3].

Студент, що не виконав вчасно лабораторну роботу, втрачає один бал за кожен тиждень затримки [3] – це формує відповідальність за терміни, що є самостійним результатом навчання (РН4.1). Водночас, якщо роботу виконано неякісно, викладач має право її не зарахувати або знизити бали – формується відповідальність за якість.

Система оцінювання як інструмент інтеграції

Системи оцінювання обох курсів побудовані за компетентнісним принципом і передбачають чотири виміри результатів навчання: знання (РН1), вміння (РН2), комунікацію (РН3) та автономність і відповідальність (РН4) [1, 3]. Подана нижче таблиця 2 зіставляє ці результати для обох курсів.

Таблиця 2. Зіставлення результатів навчання дисциплін ТП та ПТП

Вимір РН	**Результати навчання – ТП (5 сем.)**	**Результати навчання – ПТП (6 сем.)**
Знати (РН1)	РН1.1: основні поняття програмування РН1.2: методи формалізації мов РН1.3: методи моделювання предметних областей	РН1.1: суть і засоби функціональної парадигми (Haskell) РН1.2: суть і засоби логічної парадигми (Prolog) РН1.3: декларативні засоби в ООП (.NET/Java)

Вимір РН	Результати навчання – ТП (5 сем.)	Результати навчання – ПТП (6 сем.)
Вміти (РН2)	РН2.1: формалізувати мови, моделювати предметні області, аналізувати програми	РН2.1: застосовувати функціональний стиль (Haskell) РН2.2: застосовувати логічне програмування (SWI-Prolog) РН2.3: використовувати декларативні засоби ООП
Комунікація (РН3)	РН3.1: обґрунтовувати власний погляд на задачу; спілкуватися щодо специфікацій та програм	(формується при захисті лабораторних, аналізі та вибору парадигми)
Автономність (РН4)	РН4.1: самостійно організовувати роботу для досягнення результату	РН4.1: самостійно організовувати роботу, нести відповідальність за якість і терміни
Форми оцінювання	2 контр. роботи + проєктне завдання + домашні завдання + поточне оцінювання + іспит (40 б.)	3 тести + захист 3-х блоків лабораторних (Haskell, Prolog, Java/.NET) + дотримання графіку + екзамен

В обох курсах вимір РН4 (автономність і відповідальність) оцінюється через дотримання студентом власного графіку роботи – а не лише через правильність виконаних завдань. Це цілеспрямоване формування відповідальності як особистісної якості фахівця.

У курсі ТП іспит являє собою письмову роботу з чотирма питаннями: два теоретичних (РН1) та два практичних (РН1.3, РН2.1). Критерії оцінювання кожного питання включають повноту розкриття (1–4 бали), логіку викладення (2 бали) та аналітичні міркування (1–4 бали) [1] – структура, що безпосередньо відображає теоретичний характер курсу і вимагає від студента не відтворення, а осмислення матеріалу.

Роль самостійної роботи

Частка самостійної роботи значна в обох курсах: 62 години в ТП та 76 годин у ПТП. Ця диспропорція на користь ПТП є методично обґрунтованою: якщо в ТП самостійна робота супроводжує та заглиблює матеріал лекцій і практик, то в ПТП вона є основним середовищем для вправляння у

парадигмальному мисленні – написання та налагодження програм на Haskell, Prolog, Java, відпрацювання патернів кожної парадигми.

Тематичний план самостійної роботи в ТП передбачає конкретні завдання по кожній темі: аналіз визначень мов програмування, побудова граматик та синтаксичних діаграм, доведення властивостей алгебр і функцій, побудова рекурсивних програм та дослідження їх властивостей, концептуальний аналіз підходів до формальної специфікації систем та ознайомлення з принципами побудови специфікацій у підходах модельно-орієнтованого (B, Z) та алгебраїчного (RAISE) проектування [1]. Ця деталізація забезпечує змістовну неперервність між аудиторними та позааудиторними формами навчання.

Результати навчання та сформовані компетентності

Інтеграція теоретичного та парадигмального підходів забезпечує формування компетентностей, що виходять за рамки кожного окремого курсу і являють собою якісно нову здатність – системне бачення програмування як наукової і технологічної дисципліни.

Загальні компетентності

Обидва курси спільно розвивають широкий спектр загальних компетентностей. Здатність до абстрактного мислення, аналізу та синтезу формується в ТП через побудову формальних моделей і доведення властивостей програм та в ПТП – через необхідність обирати між принципово різними способами опису однієї й тієї ж задачі. Здатність до математичного формулювання та дослідження дискретних математичних моделей є безпосередньою метою ТП і передумовою ПТП. Здатність обґрунтовувати вибір методів і підходів для розв'язування теоретичних і прикладних задач у галузі комп'ютерних наук формується саме на перетині двох курсів: теорія дає критерії вибору, а парадигмальна практика – досвід застосування [1].

Фахові компетентності

На рівні фахових компетентностей особливо важлива СК18.2 – здатність до алгоритмічного та логічного мислення [3]. Ця компетентність виявляється в кожній з трьох парадигм по-різному: рекурсивне мислення та еквіваленція функцій – у функціональній; логічне виведення та стратегії пошуку – у логічній; декларативний опис залежностей та метапрограмування – в декларативно-об'єктно-орієнтованій. Проходження всіх трьох забезпечує об'ємне, багатовимірне алгоритмічне мислення.

Програмний результат навчання ПРН19.2 – здатність аналізувати, оцінювати і вибирати інструментальні та обчислювальні засоби, парадигми, технології, алгоритмічні і програмні рішення при проєктуванні та розробці

програмних систем [1, 3] – є «точкою зустрічі» обох підходів. Він потребує і теоретичного розуміння семантики та властивостей програм, і практичного досвіду роботи з різними парадигмами. Без теоретичного курсу цей результат залишається формальним – студент «знає парадигми», але не вміє обирати між ними аргументовано; без парадигмального курсу теоретик не має досвіду, де застосовувати свій апарат.

Не менш важливим є ПРН18.2 – здатність застосовувати підходи щодо оцінки та забезпечення якості програмного забезпечення [1, 3]. Саме теорія програмування надає для цього математичний апарат (доведення коректності, формальна верифікація), а парадигмальний курс – практичні засоби його реалізації (типові системи Haskell як інструмент статичного запобігання помилок; декларативні специфікації як контракти між компонентами системи).

Спостереження з педагогічної практики

Аналіз навчального процесу та захистів робіт у курсі ПТП показує якісні переваги, що виникають завдяки попередньому опануванню курсу ТП у 5-му семестрі. Спостереження за динамікою засвоєння тем фіксують такі характерні патерни студентського сприйняття:

- при вивченні Haskell вони значно природніше приймають відсутність стану і мутабельності – адже алгебра програм, з якою вони вже знайомі, описує програму як функцію, а не як послідовність змін стану. Поняття монади перестає бути «страшним» – це просто тип-обгортка з двома операціями, що відповідають певним алгебраїчним законам;

- при вивченні Prolog вони значно впевненіше орієнтуються в операційній семантиці: алгоритм уніфікації не є «магією інтерпретатора» – це конкретний алгоритм, схожий на розв'язання системи рівнянь, що вивчався ще в курсі математичної логіки. Механізм відкатів сприймається як природна реалізація пошуку в дереві виводу;

- при вивченні декларативних засобів у Java вони критично підходять до вибору між контрактним і ресурсним підходами до опису інтерфейсів, розуміючи різницю між специфікацією і реалізацією – що є прямим наслідком теоретичного розуміння різниці між синтаксисом і семантикою програм.

Водночас спостерігається і зворотний ефект: парадигмальний курс поглиблює розуміння теоретичних понять. Після практичного знайомства з монадами студент краще розуміє абстрактний зміст часткових функцій; після роботи з Prolog – яскравіше відчуває практичне значення алгоритму уніфікації.

Висновки та перспективи

Інтеграція теоретичного та парадигмального підходів у підготовці бакалаврів інформатики є не методичним прийомом, а концептуальною необхідністю, що випливає зі самої природи програмування як науки і практики. «Теорія програмування» і «Парадигми та технології програмування» є двома взаємодоповнювальними вимірами одного феномену. Перший курс формує поняттєвий фундамент: мову опису, інструменти формалізації, принципи доведення коректності – категоріальний апарат, що дозволяє говорити про програми строгою математичною мовою. Другий реалізує цей фундамент у різних обчислювальних стилях, показуючи, як одні й ті самі теоретичні конструкти – нерухомі точки, уніфікація, алгебри функцій, предметні специфікації – живуть у функціональному, логічному та декларативно-об'єктно-орієнтованому програмуванні.

Ефективна методична інтеграція забезпечується сукупністю умов, виявлених у педагогічній практиці кафедри. По-перше, логічна послідовність розміщення: теоретичний курс у 5-му семестрі передує парадигмальному у 6-му. По-друге, спільна математична база: обидва курси спираються на дискретну математику, математичну логіку та теорію алгоритмів, що утворює єдине понятійне поле. По-третє, цілеспрямоване відображення: кожне теоретичне поняття ТП знаходить конкретне програмне втілення в одній або кількох парадигмах ПТП, і цей зв'язок явно артикулюється у навчанні. По-четверте, компетентнісно орієнтована система оцінювання, що охоплює всі чотири виміри – знання, вміння, комунікацію та автономність – і є узгодженою між курсами. По-п'яте, діалогічні форми навчання, що вимагають від студента обґрунтування, а не лише відтворення.

Описаний підхід демонструє, що принцип методологічного циклу – практика → наукове дослідження → теорія → технологія → практика [2] – є не лише описом наукового методу, а й продуктивним дидактичним принципом. Студент проживає цей цикл: спочатку зустрічається з практикою програмування у попередніх курсах, потім отримує теоретичні інструменти її аналізу (ТП), потім повертається до практики на якісно новому рівні (ПТП), розуміючи, чому та чи інша технологія працює саме так.

Перспективою подальшого розвитку описаного підходу є розробка міждисциплінарних проєктних завдань, де студент самостійно обирає парадигму та обґрунтовує вибір формальними аргументами, що дає змогу вивести інтеграцію підходів на рівень реального дослідницького мислення.

Підсумовуючи, можна стверджувати: бакалавр інформатики, що пройшов обидва курси в їхньому методичному взаємозв'язку, отримує не

просто набір знань і навичок, а цілісний фаховий світогляд – здатність бачити в кожній програмі теоретичний об'єкт зі строго визначеними властивостями і водночас – результат усвідомленого парадигмального вибору.

Список використаних джерел

1. Шишацька О. В., Криволап А. В. Робоча програма навчальної дисципліни «Теорія програмування» для студентів спеціальності 122 «Комп'ютерні науки», ОПП «Інформатика». – Київ : Київський національний університет імені Тараса Шевченка, 2025. – 18 с.
2. Нікітченко М. С. Теорія програмування : навчальний посібник. – Київ : Видавничо-поліграфічний центр «Київський університет», 2023.
3. Шишацька О. В. Робоча програма навчальної дисципліни «Парадигми та технології програмування» для студентів спеціальності 122 «Комп'ютерні науки», ОПП «Інформатика». – Київ : Київський національний університет імені Тараса Шевченка, 2026. – 20 с.
4. Стандарт вищої освіти за спеціальністю 122 «Комп'ютерні науки» для першого (бакалаврського) рівня вищої освіти / Міністерство освіти і науки України. – Київ, 2019.
5. Hoare C. A. R. An Axiomatic Basis for Computer Programming // Communications of the ACM. – 1969. – Vol. 12, No. 10. – P. 576–580.
6. Floyd R. W. The Paradigms of Programming // Communications of the ACM. – 1979. – Vol. 22, No. 8. – P. 455–460.
7. Редько В. Н. Композиційні бази даних і програмування. – Київ : Наукова думка, 1992.
8. Liskov B. and Guttag J. Abstraction and Specification in Program Development. Cambridge, MA, USA: MIT Press, 1986.