Formation of software design skills among software engineering students

Andrii M. Striuk[0000−0001−9240−1976]

Kryvyi Rih National University,
11 Vitalii Matusevych Str., Kryvyi Rih, 50027, Ukraine
andrey.n.stryuk@gmail.com
http://mpz.knu.edu.ua/pro-kafedru/vikladachi/224-andrii-striuk

Abstract. The study focuses on one of the mobile-oriented environment competence components for software engineering (SE) students. It has been demonstrated that the implementation of the higher education standard for SE bachelors has generated a number of issues in terms of ensuring training quality, principally due to a lack of specification for both skills and learning outcomes. Designing a precise framework of professional competencies for SE bachelors is one method to overcome these issues. The research examines methods for developing K14 (the ability to participate in software design, including modeling (formal description) of its structure, behavior, and working processes), a critical particular professional competency for future software engineers. Recommendations for software design teaching techniques, learning content, modeling and design tools, and assessment of the level of formation of relevant competence are developed based on a historical and genetic review of software design training among SE students in the UK, USA, Canada, Australia, New Zealand, and Singapore. The industrial-style software design training (studio training) is used as an example. The transition from architectural to detailed design, as well as project implementation, are discussed. The study's future prospects include substantiating the third engineering component of SE – software construction (after requirements engineering and design engineering).

Keywords: software engineering · software design · professional training of software engineering students · software design skills SE bachelors

1 Вступ

Затвердження у 2018 році стандарту вищої освіти за спеціальністю 121 «Інженерія програмного забезпечення» для першого (бакалаврського) рівня вищої освіти [11] у світлі нової процедури акредитації освітніх програм породило дві основні проблеми, розв'язання яких було покладено на гарантів освітніх програм:

1. Стандарт надає надзвичайно високий ступінь свободи у трактуванні змісту компетентностей та програмних результатів навчання, що призводить до появи суттєво різних та негармонізованих освітніх програм

і планів, оцінка яких виконується експертами НАЗЯВО за також розмитими критеріями якості. При цьому аналогічні зарубіжні стандарти (університетські, державні та світові) мають високий рівень деталізації складових компетентностей та критеріїв оцінювання їх сформованості – так само, як у США, Австралії та Японії критерії якості освітніх програм є деталізованими для певної галузі освіти, а не застосовуваними до усіх. Відсутність чітких критерії оцінювання якості, специфічних для галузі знань, породжує ризики порушення принципу академічної доброчесності при їх оцінюванні.

2. Висока швидкість зміни змісту технологічної складової підготовки фахівців з інженерії програмного забезпечення поряд із намаганням укладачів освітніх програм якщо не випередити, то хоча б відповісти на вимоги виробництва, призводить до інструментального ухилу в навчанні інженерії програмного забезпечення – аж до нівелювання найважливіших для сталого професійного розвитку інженера-програміста загальних компетентностей. Такий ухил призводить до нерозрізненності інженерії програмного забезпечення від інших спеціальностей галузі знань 12 «Інформаційні технології» та професійної дезорієнтації вступників та студентів за нею.

Розв'язання поставлених проблем вимагає системного опанування світового досвіду підготовки фахівців з інженерії програмного забезпечення (ІПЗ) у його генезисі [17, 18] та відповідного проектування системи професійних компетентностей (змісту, показників сформованості та засобів діагностики), як загальних [16], так й спеціальних [19].

Серед спеціальних компетентностей фахівця з ІПЗ на особливу увагу заслуговують ті з них, що відображають «сутність та дух» ІПЗ – специфіку професійної діяльності інженерів-програмістів, що не може бути набута при навчанні за іншими спеціальностями галузі знань 12 «Інформаційні технології». Так, рекомендації до розробки навчальних програм для бакалаврів ІПЗ визначають компетентність з пошуку компромісів, сутність якої полягає в узгодженні суперечливих цілей проекту, пошуку прийнятних компромісів за обмежень вартості, часу, знань, існуючих систем і організацій: «студенти повинні займатися проблемами, які приводять їх до суперечливих і мінливих вимог. ... Компоненти навчальної програми повинні спрямовувати на такі проблеми з метою забезпечення високоякісних функціональних і нефункціональних вимог та розробку якісного програмного забезпечення» [9, с. 21].

Пошук компромісів та узгодження протиріч є традиційною діяльністю з інженерного проектування (дизайну), що не обов'язковою для всіх фахівців з інформаційних технологій, проте є ключовою для інженерів з програмного забезпечення. У стандарті [11] їй відповідає спеціальна компетентність К14 – «здатність брати участь у проектуванні програмного забезпечення, включаючи проведення моделювання (формальний опис) його структури, поведінки та процесів функціонування». Формування здатності до проектування програмного забезпечення є визначальним у тому, чи дійсно є випускник освітньої програми з ІПЗ інженером-програмістом.

Огляд освітніх програм з ІПЗ, доступних на сайтах ЗВО України, показує, що навчання проектування програмного забезпечення виконується за трьома основними напрямами:

1) штучне уведення елементів проектування до різних курсів у тісному зв'язку з інструментальними засобами на прикладі розв'язання типових спрощених задач певної галузі;
2) спонтанне навчання елементів проектування під час виробничої практики на реальних задачах;
3) навчання засобу моделювання програмного забезпечення (зазвичай, UML).

Лише у незначній кількості освітніх програм спостерігаються спроби урахування світового досвіду організації навчання проектування на основі синергетичного об'єднання академічних та промислових форм і методів навчання.

Тому **метою** нашої статті є історико-генетичний огляд практики навчання проектування програмного забезпечення майбутніх фахівців з ІПЗ.

2 Навчання проектування програмного забезпечення: зарубіжний досвід

Виокремлену у стандарті [11] спеціальну компетентність К14 (здатність брати участь у проектуванні програмного забезпечення, включаючи проведення моделювання (формальний опис) його структури, поведінки та процесів функціонування) В. М. Пелевін співвідносить із здатністю та готовністю до здійснення системотехнічного проектування, що включає проектування системного, прикладного ПЗ та мереж ЕОМ [13, с. 14].

Моделювання та аналіз можна вважати основними поняттями будь-якої інженерної дисципліни, оскільки вони є важливими для документування та оцінки проектних рішень і альтернатив [9, с. 31]. Проектування програмного забезпечення стосується питань, методів, стратегій, способів подання та шаблонів, що використовуються для визначення способу реалізації компонента або системи [9, с. 32].

Д. Керрінгтон (David Carrington) [2] визначає, що проектування ПЗ – це етап розробки ПЗ, під час якого специфікація перетворюється в структуру, придатну для реалізації. Навчання проектуванню повинно охоплювати як об'єкт проектування, так і процес, за допомогою якого цей об'єкт створюється [2, с. 547].

Ч. Ху (Chenglie Hu) [6] ставить ряд питань стосовно того, чим є проектування програмного забезпечення:

1. *Чи є проектування програмного забезпечення інженерною діяльністю?* «Наприклад, проектування мосту повинно здійснюватись відповідно до фізичних законів, але при проектуванні програмного забезпечення немає законів, яких слід дотримуватися, принаймні теоретично. Архітектура

моста добре помітна неозброєним оком, тоді як програмне забезпечення – абстрактна сутність, над якою ми працюємо лише з різними його відображеннями. На практиці інженерне проектування має передувати офіційному будівництву мосту, тоді як будівництво програмного забезпечення може відбуватися без чіткого проекту. Однак найсерйозніша відмінність полягає в тому, що після початку будівництва зміни в специфікації інженерного продукту можуть бути недопустимі, тоді як зміни вимог до програмного забезпечення, як правило, очікуються і дійсно можуть відбутися в будь-який час протягом життєвого циклу програмного забезпечення» [6, с. 63].

2. *Чи можна вважати саму комп'ютерну програму твором мистецтва?* «Ймовірно, це можливо, але не завжди. ... Причина в тому, що проектувальник може, наприклад, через програмні обмеженні різної природи порушити деякі принципи проектування для розв'язання нагальних проблем. Проектування – це баланс між розширюваністю, дотриманням принципів проектування та урахуванням програмних обмежень, а отже, може бути елегантним та художньо привабливим» [6, с. 63].
3. *Чи можна проектування програмного забезпечення визначити через проектну діяльність?* «Проектна діяльність різноманітна; багато аспектів, безумовно, є технічними, але деякі можуть бути і соціальними, але жодні з них, ймовірно, не будуть стандартизовані для характеристики проекту або його реалізації» [6, с. 63].
4. *Чи можна визначити проектування програмного забезпечення за допомогою артефактів чи етапів проектування?* «Стандарт IEEE 1016-2009 для опису проекту програмного забезпечення визначає зміст та процедуру його опису, ... однак спосіб опису проекту ... може суттєво варіюватись ..., а коли професіонали не мають консенсусу що артефактів проектування, вони, ймовірно, не матимуть консенсусу й щодо етапів проектування. При цьому практики гнучких методів вірять, що проектування не лише високо ітеративне, але й емерджентне, моделі часто помилкові, і лише кодування, виконання тестів та рефакторинг коду відкривають правду про проект» [6, с. 63–64].

У результаті Ч. Ху визначає *проектування програмного забезпечення* як систематичний розумовий процес, у якому проектувальники генерують, оцінюють та конкретизують концепції програмної системи, структура та функції якої відповідають цілям клієнтів або потребам користувачів, задовольняючи при цьому певний набір обмежень [6, с. 64].

Процес проектування водночас і надзвичайно креативний, і надзвичайно складний, тому не дивно, що по-справжньому видатних проектувальників (дизайнерів) мало. Однак потреби в нових проектах дуже високі, і щоб їх задовільнити, в рамках кожної архітектурної та інженерної галузі розроблено набори принципів, що надають можливість проектувальникам із середніми здібностями створювати проектні рішення прийнятної якості. Хоча ІПЗ значно молодша за інші інженерні галузі, за час її існування також накопичені певні знання про те, як створювати проекти, і майбутні інженери-

програмісти повинні оволодіти цими знаннями, тому навчання проектування має бути невід'ємною частиною їхньої професійної підготовки [15].

«Виняткові люди народжуються з відчуттям того, як робити гарні проекти. Більшість з нас повинні побачити приклади хороших проектів, перш ніж ми зможемо самостійно придумати гарний проект. Тому ми навчаємо проектуванню на прикладах, надаючи студентам ескізи проектів, а потім багато відгуків про їх спроби вдосконалити проект» [7, с. 33].

Р. М. Грем (Robert M. Graham) у статті 1970 року [4] пропонує методику навчання проектування, за якої уводиться єдина ключова ідея та досліджуються її наслідки в середовищі, яке свідомо ігнорує питання ефективності. Після повного розвитку логічних наслідків цієї ідеї, додаткові ключові ідеї об'єднуються одна за одною та досліджуються їх наслідки. Далі розглядаються ефективність та інші реальні обмеження, досліджуючи наслідки кожного з них. Цю процедуру продовжують, доки не буде досягнуто реалістичного проекту. Крім того, для ілюстрації розробки проекту використовується спеціально розроблений та задокументований кейс.

Д. М. Вайс (David M. Weiss) у [22] представляє більш пізній (середина 1980-х рр.) досвід навчання проектування програмного забезпечення, інваріантного до застосовуваної методології проектування. Його підхід передбачає два етапи: спочатку студенти знайомляться з принципами, що лежать в основі методології, та досвідом застосування цих принципів до невеликих чітко визначених задач. На другому етапі студентам пропонується реальна задача, яку вони мають вирішити самостійно в змодельованому робочому середовищі, отримуючи вказівки лише з методологічних питань. Перший етап представлений у вигляді циклу лекцій. Другий – контролюється експертом з проектування. В результаті першого етапу студенти отримують певне розуміння принципів, що лежать в основі методики проектування, бачать зразки застосування цих принципів та практикуються у їх застосуванні на навчальних прикладах. Після першого етапу студенти здатні використовувати методики проектування під керівництвом досвідченого проектувальника у процесі його майже щоденної взаємодії зі студентами. В результаті другого етапу студенти набувають достатньо досвіду з проектування, щоб здійснювати його з меншими настановами, спілкуючись із досвідченим проектувальником приблизно раз на тиждень [22, с. 1156].

На установчій зустрічі проекту між студентами розподіляються ролі, які вони будуть виконувати під час проекту. Студенти знайомляться з послідовністю основних етапів проекту, отримують перелік документів, які потрібно було пред'явити, та порядок їх підготовки. Також студентам надаються короткі пам'ятки щодо процедури керування конфігурацією, опис відповідальності та складу команди з контролю якості. Кожен студент повинен розробити та задокументувати принаймні один модуль системи [22, с. 1157]. Д. М. Вайс відзначає, що студенти залишались невдоволеними тим, що були зосереджені на проектуванні, але так і не розробили готовий продукт через складність проектованої задачі (однак більш проста задача не надала б можливості продемонструвати та засвоїти всі необхідні прийоми та методи

проектування). Загалом, студенти відчували, що вони засвоїли методологію проектування, але були розчаровані тим, що не створили жодного рядка коду. Щоб подолати це розчарування, можна запропонувати студентам продовжити роботу над реалізацією проекта в наступному семестрі [22, с. 1158].

Д. М. Вайс надає наступні рекомендації з організації навчання проектування ПЗ [22, с. 1159]:

1) консультувати студентів має фахівець з проектування, який може забезпечити неухильне дотримання методології;
2) задача з проектування має бути достатньо складною, і студентам заборонено її переспрощувати;
3) студентів потрібно спонукати до прийняття власних проектних рішень: жоден із викладачів не має втручатись у процес прийняття рішень;
4) розмір студентської групи має бути невеликим, щоб дозволити викладачам уважно стежити за успіхами кожного студента.

Е. Ламм (Ehud Lamm) [10] зазначає, що задачею курсу з проектування програмного забезпечення є розвиток навичок проектування програм через викладання основних концепцій інженерії програмного забезпечення, необхідних для вивчення та аналізу альтернативних проектів програмного забезпечення.

Computing Curricula 2020 [3, с. 120] визначає наступні компетенції, що відносяться до проектування програмного забезпечення:

1. Здатність представити особам, що приймають рішення у бізнесі, архітектурно значущі вимоги з документа із їх специфікаціями.
2. Здатність оцінити та порівняти компроміси з альтернативних варіантів проекту для задоволення функціональних і нефункціональних вимог та написати коротку пропозицію, в якій узагальнюються ключові висновки для клієнта.
3. Здатність розробити високорівневий проект конкретної підсистеми, зрозумілий для нефахівців з проектування, враховуючи архітектурні шаблони та шаблони проектування.
4. Здатність розробити для клієнта високорівневий проект конкретної підсистеми, використовуючи принципи проектування та наскрізні аспекти для задоволення функціональних та нефункціональних вимог.
5. Здатність до оцінки складників тестування якості програмного забезпечення при проектуванні підсистем та модулів для розробника / виробника.
6. Здатність створювати документацію із проектування програмного забезпечення для подальшої ефективної роботи аналітиків, розробників, тестувальників, фахівців із підтримки та ін.

Ядро знань щодо проектування програмного забезпечення Ч. Ху [6] пропонує розділити на 2 категорії: знання про процес проектування та знання про технології проектування – до останніх він включає й шаблони проектування. Знання про процес автор визначає як знання про загальновизнані

етапи проектування, а також парадигми або методики проектування, що використовуються, зокрема, для створення артефактів проектування [6, с. 65]. Аналіз вимог до програмного забезпечення є частиною процесу архітектурного проектування, що охоплює функціональні та поведінкові аспекти архітектури. Архітектурний проект стосується не тільки того, що є системою, але і того, що робить система. Наступний етап проектування – неархітектурне (детальне) проектування – стосується модуляризації та деталізації інтерфейсів елементів проекту, їх алгоритмів та процедур, а також типів даних, необхідних для підтримки архітектури та задоволення вимог. Неархітектурний проект визначає функціональність та структуру програмного забезпечення на високому рівні деталізації, проте недостатньому для реалізації. Процес проектування також передбачає управління виробництвом проектних артефактів для забезпечення правильної та точної реалізації дизайнерських ідей та рішень. Ч. Ху вважає, що процес проектування не є завершеним, якщо проект не реалізований у коді [6, с. 66].

К. Пірс (Keith Pierce), Л. Денін (Linda Deneen), Г. Шут (Gary Shute) [15] уважають, що ключовим питанням навчання проектування є раціональний вибір серед багатьох альтернативних рішень [15, с. 220]. «Нам здається, що викладання проектування значно покращиться за ... [умов]: представлення альтернативних стратегій проектування та альтернативних рішень, отриманих за допомогою цих стратегій; представлення критерії оцінки якості альтернатив та спонукання студентів до виконання завдань, під час яких вони могли б зробити вибір альтернативи та обґрунтувати його» [15, с. 220–221]. Методи проектування автори розділяють на низькорівневі (застосовуються для проектування малих модулів – вибір упорядкованими та невпорядкованими масивами, між рекурсією та ітерацією тощо) та високорівневі (застосовуються для прийняття рішень про організацію програмних модулів).

Й. Джа (Yanxia Jia), Й. Дао (Yonglei Tao) [8] вважають моделювання центральною складовою процесу розробки якісного програмного забезпечення. Викладаючи проектування програмного забезпечення, викладачі мають робити наголос на створенні відповідної моделі для обмірковування проблеми та на використанні властивостей моделі для конструювання рішення [8, с. 702].

Автори [8] визначають такі ключові концепції:

- *моделювання* – процес створення абстрактного, графічного або математичного опису задачі проектування, під час якого розробник замінює складну та детальну реальну ситуацію зрозумілою моделлю, що відображає сутність задачі;
- *еволюція моделі* в ітераційному процесі розробки має тенденцію до збереження форми представлення, тоді як *трансформація моделі* передбачає зміну точки зору, з якої розглядається задача проектування, та зміну структури моделі проектування;
- *рефакторинг коду* – це процес зміни коду комп'ютерної програми для того, щоб зберегти її здатність до розвитку, покращити її читабельність або спростити структуру, зберігаючи при цьому існуючу функціональ-

ність. Навчання рефакторингу не лише дає студентам практичні навички програмування, а й допомагає їм зрозуміти найважливіші принципи ІПЗ;
– *шаблони проектування* можуть допомогти розробникам вирішити певні задачі проектування, а також покращити існуючі проекти.

Перепроектування програмного забезпечення для кращого повторного використання чи набуття інших якостей є ілюстрацією трансформації моделі. Поетапна трансформація моделі вимагає від студентів постійної оцінки розриву між тим, що зроблено, і тим, що необхідно зробити. Така діяльність особливо корисна для формування у студентів здатності до оцінки існуючого рішення та аналізу зисків і витрат [8, с. 705].

Шаблони проектування програмного забезпечення часто використовуються для вирішення поширеної задачі шляхом «загального» підходу до неї. Й. ван Нікерк (Johan van Niekerk), Л. Футчер (Lynn Futcher) [12] вказують, що шаблон проектування надає концептуальну модель рішення з найкращих практик, яка, у свою чергу, використовується розробниками для створення конкретної реалізації їх задачі. Використання шаблонів проектування надає ряд переваг: 1) шаблон надає орієнтир щодо найкращої практики; 2) використання шаблону надає розробникам спільний «словник» для легкого та чіткого обговорення складних концепцій проектування. Завдяки цим та іншим перевагам шаблони проектування часто вивчаються на курсах проектування програмного забезпечення [12, с. 75].

Шаблон можна описати як «контекстне рішення задачі»:

– контекст – це повторювана ситуація, в якій застосовується шаблон;
– задача стосується мети, що має бути досягнута у цьому контексті, але вона також стосується будь-яких обмежень, що виникають у контексті;
– задача повинна бути повторюваною;
– рішення забезпечує загальний проект (основне рішення), яке визначає сутність рішення задачі з урахуванням даного контексту та обмежень [12, с. 77].

Шаблони проектування надають проектувальникам програмного забезпечення три основні переваги:

– по-перше, відомо, що рішення є надійним, оскільки воно перевірено часом;
– по-друге, переваги та недоліки шаблону відомі заздалегідь, і вони можуть бути враховані під час ескізного проектування;
– по-третє, шаблони утворюють загальний словник, який може полегшити спілкування між різними зацікавленими сторонами [12, с. 78].

Ч. Вільямс (Chad Williams) та С. Курковскі (Stan Kurkovsky) [24] підкреслюють, що процес навчання застосування доцільних шаблонів проектування програмного забезпечення можна зробити творчим шляхом перетворення самого процесу проектування на конструктивістське навчальне середовище.

Для цього вони пропонують не обмежувати тематику студентських проектів та створювати умови для виходу студентів із зони комфорту шляхом застосування нових апаратних платформ та інтерфейсів до них. Проміжна оцінка проекту виконувалась за рівнем доцільності та кількістю застосованих шаблонів проектування, а заключна – за рівнем креативності усього проекту.

Фреймворк можна визначити як напівзавершену програму, що містить певні фіксовані складові, спільні для всіх програм у проблемній області, поряд з певними змінними складовими, унікальними для кожної програми, створеної з нього. Більшість комерційного програмного забезпечення розробляється за допомогою фреймворків шляхом розширення та налаштування стандартних загальних функцій, які вони надають. З. Алі (Zoya Ali), Дж. Болінгер (Joseph Bolinger), М. Герольд (Michael Herold), Т. Лінч (Thomas Lynch), Дж. Раманатан (Jay Ramanathan), Р. Рамнат (Rajiv Ramnath) [1] вказують, що розробники, які володіють принципами об'єктно-орієнтованого проектування, не застосовують при розробці програмного забезпечення із використання певного фреймворку, зосереджуючись на тому, щоб просто «змусити його працювати». Адаптація до нового фреймворку може стати викликом для розробників-початківців через те, що застосування шаблонів проєктування у новому фреймворку може призвести до поганого проектування та неправильного використання фреймворку. Автори [1] пропонують триступеневий процес навчання проектування із використанням фреймворків, спрямований на подолання вказаної проблеми:

1. Спочатку студентам пропонується спроектувати їх програму за допомогою об'єктно-орієнтованого проектування. Використовуючи формулювання задачі, студенти спонукаються до її «об'єктно-орієнтованого обмірковування» та використання попереднього досвіду для створення об'єктів, класів, обов'язків, взаємозв'язків, методів та інших сутностей UML.
2. Далі студентам пропонується переробити програму за допомогою шаблонів проектування. Одним з таких шаблонів є шаблон «модель – представлення – контролер» (MVC). Ідея цієї моделі полягає в тому, щоб ізолювати логіку домену програми від способу подання даних користувачеві (тобто інтерфейсу користувача програми), щоб ці два дуже важливі компоненти будь-якої програми могли бути розроблені, реалізовані та підтримувані окремо.
3. Нарешті, студентам пропонується скоригувати використання шаблону проектування відповідно до обраного фреймворку.

Для кращого розуміння фреймворку та безшовної інтеграції проекту із фреймворком студентів необхідно навчити шаблонів проектування, що диктуються фреймворком, та порівняти їх зі стандартним шаблоном проектування. «Як природне доповнення до шаблонів проектування, UML є мовою спілкування студентів та викладачів» [21, с. 42]

Викладання UML стимулює до застосування низхідного підходу до ІПЗ: спочатку студентів навчають виявленню вимог до програмного забезпечен-

ня, а потім перетворенню вимоги на архітектуру програмного забезпечення, низькорівневий дизайн та, нарешті, реалізацію. UML часто використовується як «lingua franca» на всіх етапах життєвого циклу, тому велика увага приділяється тому, щоб студенти створювали високоякісні, синтаксично та семантично правильні діаграми UML [23, с. 19].

Автори [1] пропонують методику, яка надасть студентами можливість скористатися перевагами фреймворку у реалізації їх проекту:

1. Перефразуйте задачу та визначить у цьому формулюванні всі іменники та дієслова. Іменники є кандидатами у об'єкти, класи та атрибути, а дієслова – обов'язками.
2. Об'єднайте виділені іменники в класи. Для цього може знадобитися відкинути нерелевантні або синонімічні іменники.
3. Об'єднайте виділені дієслова у класи, екземпляри та обов'язки.
4. Призначте обов'язки, визначивши необхідні методи для їх виконання.
5. Пройдіться по сценарію, щоб переконатися, що кожен сценарій підтримується методами, та визначте взаємозв'язки між ними [1, с. S3G-3–S3G-4].

Проектування за своєю суттю є міждисциплінарним предметом, на який суттєво впливає людський фактор, тому викладання проектування – це не лише процес навчання самого проектування, а й процес подальшого розвитку соціальних навичок студентів [6, с. 70]. С. Яржабек (Stanislaw Jarzabek) [7] зазначає, що саме командно зорієнтовані проектні курси створюють можливість навчати принципів ІПЗ, коли їх застосування дійсно необхідно та вигідно, та пропонує наступну їх класифікацію:

(1) виробнича практика: студенти працюють над реальними задачами у промисловому середовищі;
(2) проектні курси, в яких студенти працюють над проблемами в різних предметних галузях під керівництвом викладачів, що є експертами у них. Іноді такі курси будуються на реальних задачах з промисловості;
(3) проектні курси, на яких студенти навчаються передових принципів проектування ПЗ та їх застосуванню у власних проектах. Оскільки для забезпечення зворотного зв'язку зі студентами викладачам необхідно детально вивчити артефакти проектування, такі проекти повинні контролюватися викладачами, що спеціалізуються на ІПЗ та добре знають задачі, над якими працюють студенти;
(4) проекти, розроблені з нуля, на відміну від проектів, в яких студенти розширюють існуюче програмне забезпечення;
(5) проекти на основі певної програмної платформи, такої як .NET, JEE, сервісу, мобільного пристрою або Facebook [7, с. 31–32].

Навички командної роботи, комунікації та письма можна розвивати у всіх вищезазначених проектних курсах. Інші навички досить складно реалізувати в рамках одного проектного курсу. Наприклад, мета проектних курсах (1) та (2) типів мета – показати студентам реальність нечітких, невизначених та мінливих вимог. Нечіткі вимоги та проектування програмного

забезпечення – це не тільки дві типові ознаки розробки програмного забезпечення, а й найбільш значні та важкі його проблеми, які важко викладати в рамках одного курсу. Проектні курси, які знайомлять студентів з нечіткими та мінливими вимогами, як правило, менш структуровані та суворі, ніж курси, які навчають студентів застосуванню принципів проектування. У навчанні застосування принципів проектування (проектний курс (3) типу) студентам необхідно надавати зразки ескізів проектів та багато детальних відгуків про їх початкові спроби вдосконалити проект. Для ефективного керівництва студентами керівники повинні бути добре знайомі з проблемною галуззю та проектними рішеннями, що може бути досить складно у проектних курсах (1) та (2) типів [7, с. 32].

Студенти пишуть підсумковий звіт, у якому вони документують плани проектів, процес розробки, архітектурні та детальні (неархітектурні) проектні рішення. Кожній команді дається одна година на презентацію своєї роботи. Викладачі оцінюють студентів на основі якості проектування, здатності оцінювати проектні рішення та аргументувати свій вибір з огляду на заявлені атрибути якості – повторне використання, розширюваність та ефективність стратегії оцінки запитів [7, с. 37–38].

Навчання на прикладі з активним зворотним зв'язком – ефективний спосіб привчити студентів до проектування у великому масштабі. Приклади, що надаються студентам, можуть включати ескізи архітектури програмного забезпечення, специфікації API, а також ілюстрації того, як застосовувати методи проектування. Деякі студенти чітко слідують конкретному прикладу для створення власних проектних рішень, у той час як інші вчаться на прикладах, але потім впроваджують інновації, експериментують з ідеями та пропонують власні варіанти методів проектування. В обох випадках найважливіше, щоб студенти чітко розуміли сутність методики проектування. Хоча лекції висвітлюють теорію принципів проектування, спілкування між командами та викладачами веде до їх розуміння. На початку навчання студенти особливо часто помиляються та потребують багато відгуків та конструктивних дискусій для виконання правильного проектування [7, с. 38].

Д. А. Тамбуррі (Damian A. Tamburri), М. Разавіан (Maryam Razavian), П. Лаго (Patricia Lago) [20] помітили, що такий підхід допомагає студентам у опануванні основних проблем проектування програмного забезпечення: а) відповідальних та раціональних проектних рішень – студенти вчаться міркувати та дійсно відповідати за власні проектні рішення; б) спільного проектування – студенти можуть конструктивно дискутувати з командами-«опонентами», досягаючи глибшого розуміння ролі проектувальника; в) ітеративного проектування – студенти навчаються на помилках та рішеннях інших людей, активно переглядаючи власні проекти; г) «соціального» проектування – студенти вчаться ефективно працювати в команді та справлятися з критичними відгуками, спричиненими різним досвідом та баченням (а отже, й надаючи відгуки з дуже різних точок зору) [20, с. 61–62].

Ж. Л. Мюртаг (Jeanne L. Murtagh), Дж. Е. Гамільтон-мол. (John A. Hamilton, Jr.) [5] описують організацію проектно-орієнтованого навчання, визначаючі наступні бажані результатів навчання студентів:

1. Студенти повинні використовувати знання з попередніх інформатичних курсів для розробки помірно складних комп'ютерних проектів, виходячи з чітких, послідовних та обґрунтовано повних вимог, представлених викладачем у проектному завданні.
2. Студенти повинні розробити високорівневі (архітектурні) проекти програм та їх інтерфейсів й отримати схвалення викладача, перш ніж переходити до детального проектування. Студенти повинні продемонструвати, що всі вимоги з проектного завдання були розподілені по певних частинах архітектурного проектування.
3. Студенти повинні розробити детальний (неархітектурний) проект програмного забезпечення та всіх інтерфейсів і отримати схвалення викладача, перш ніж приступати до реалізації (тобто написання коду).
4. Студенти повинні розробити план випробувань для кожного проекту. План випробувань повинен демонструвати, як будуть перевірені всі вимоги до програмного забезпечення, а також містити графік тестування програмного забезпечення.
5. Студенти повинні розробити документацію, яка показує, наскільки їхнє програмне забезпечення та план випробувань відповідають вимогам стандарту IEEE/EIA 12207 [5, с. 5.577.2–5.577.3].

«Існує цікава паралель між викладанням та навчанням, з одного боку, та шаблонами проектування, з іншого. Шаблони проектування – це не лише гарна практика: вони є кульмінацією випробуваних та перевірених методів проектування програмного забезпечення, які надають такі бажані властивості, як гнучкість та повторне використання. У кількох випадках, безперечно, як й інші проектувальники програмного забезпечення, ми розв'язували задачу проектування, і лише пізніше дізнавалися, що насправді просто застосували певний шаблон. Це заспокоює, але справа в тому, що ми підсвідомо застосували те, що прийнято вважати гарною практикою. З викладанням ми часто робимо те саме; ми несвідомо використовуємо методику, коріння якої – в усталеній теорії навчання. Однак в обох випадках застосування перевірених методів важливо для отримання якісних результатів» [21, с. 40].

Я. Уоррен (Ian Warren) [21] методично обґрунтував програмні результати навчання проектування програмного забезпечення:

1. *Визначення та опис цілей проектування програмного забезпечення.* Цілі проектування включають коректність, надійність, гнучкість, багаторазове використання та ефективність. Студенти повинні враховувати, що програмне забезпечення має бути не лише правильним, але й задовольняти останні чотири нефункціональні цілі.
2. *Інтерпретація та конструювання UML-моделей програмного забезпечення.* UML, будучи стандартною галузевою нотацією, є очевидним вибором для навчання проектування, тому студенти повинні вміти читати та записувати моделі мовою UML.

3. *Пояснення поняття шаблону проектування та опис підмножину шаблонів.* Шаблони проектування втілюють перевірені дизайнерські рішення. Студенти повинні зрозуміти, що саме використання шаблонів, а не намагання вирішити задачу «з нуля», є інженерним підходом.
4. *Застосування шаблонів з адаптацією їх до вирішення реальних задач.* Поінформованість про шаблони є важливою, але не замінює здатності застосовувати шаблон для вирішення конкретної задачі. Цей програмний результат стосується глибших, функціональних знань.
5. *Застосовувати нещодавно набутих та розвинених навичок програмування.* Студенти-початківці мають гарні знання про основи ООП, але обмежені знання про бібліотеки класів Java та більш глибокі аспекти програмування. Цей програмний результат спрямований на озброєння студентів навичками реалізації програмних проектів.
6. *Робота з відносно великими проектами програмного забезпечення.* Так як досвід студентів у розробці програмного забезпечення, як правило, обмежений невеликими програмами, що включають кілька класів, ознайомлення студентів з великими проектами – це гарна підготовка до роботи над заключним проектом останнього року навчання та виробничої практики [21, с. 41-42].

Я. Уоррен виокремив методи, ефективні для навчання проектування програмного забезпечення:

– *проблемні запитання виду «що буде, якщо ...?»*. Я. Уоррен наводить приклад використання таких запитань під час розгляду діаграм класів UML і, зокрема, відносин та множинних обмежень: «Представляючи діаграму класів, ми запитуємо студентів, що насправді означає обмеження множинності? Їм доводиться інтерпретувати діаграми та оцінювати власне розуміння, а не пасивно сприймати теорію. Коли ми бачимо, що студенти зрозуміли, ми можемо змінити обмеження; незначні зміни на діаграмі можуть мати значно вплинути на сутність моделі» [21, с. 43];
– *рольові ігри*: «при розгляді основних ідей об'єктно-орієнтованого підходу студентів діють як об'єкти та відтворюють сценарії, що показують, як між об'єктами формуються динамічні зв'язки та як опрацьовуються повідомлення... Рольова гра природно призводить до документування сценаріїв за допомогою UML – щойно студенти відпрацюють сценарій, вони документують його за допомогою діаграми взаємодії об'єктів UML. На основі початкової діаграми класів та діаграми взаємодії, створеної з рольової гри, ми можемо досліджувати питання добре сформованих моделей, де різні точки зору в кінцевому підсумку повинні бути взаємно узгодженими» [21, с. 43];
– *активне навчання*, від уведення діяльнісних фрагментів під час лекцій до сеансів інтерактивної та проблемно-орієнтованої діяльності як засобу набуття студентами функціональних знань щодо шаблонів проектування [21, с. 48];
– *взаємонавчання* (peer-learning) проявляється в тому, що студенти мають офіційного партнера для навчання, з яким вони парно програмують та

співпрацюють у вирішенні курсових завдань. Крім того, студенти працюють у малих групах над розв'язанням задач в аудиторії [21, с. 48].

Дж. Уіттл (Jon Whittle), К. Н. Булл (Christopher N. Bull), Дж. Лі (Jaejoon Lee), Дж. Котонья (Gerald Kotonya) [23] пропонують ще одну альтернативу традиційному навчанню – студійну освіту (studio-based education), що виводить на перше місце рефлексивну практику як спосіб розвитку навичок проектування. У студії студенти запрошуються до критичного обмірковування власного та чужого дизайну, для цього використовуються різноманітні методи, такі як наставництво, критика дизайну, взаємонаставництво та узгоджене оцінювання. Студійний курс зазвичай, але не завжди, викладається у приміщенні, призначеному для цієї мети, тобто в студії. Фізична студія розглядається як ключовий елемент, оскільки вона дозволяє студентам постійно демонструвати власну роботу, що з часом заохочує рефлексивну практику. Студія також заохочує часті та неформальні взаємодії між студентами, що призводить до взаємонавчання [23, с. 12].

У табл. 1 наведено порівняльну характеристику середовища студійного та традиційного навчання.

При реалізації студійного курсу в Університеті Ланкастеру автори [23] виявили, що за студійного навчання студенти майже не використовували формальне моделювання (моделювання з використанням формалізованого опису, такого як UML, де використовуються точні позначення та правильний синтаксис), застосовуючи натомість неформальне моделювання з використанням або спеціальної нотації (наприклад, малювання фігур на дошці (рис. 1), ручкою на папері чи за допомогою інструменту цифрового ескізування), або вільної інтерпретації визнаної мови моделювання (наприклад, малювання діаграми класів UML без дотримання правильного синтаксису) [23, с. 16]. Хоча студенти в студії мало займалися формальним моделюванням, вони регулярно придумували неформальні моделі. Сюди входять як моделі UML (як правило, варіанти використання та діаграми класів), ескізи (наприклад, архітектура високого рівня, ескізи дизайну інтерфейсу користувача), так і моделі на основі процесів (наприклад, діаграми вигоряння, які використовуються в гнучкій розробці) [23, с. 17].

За студійного підходу студенти займалися моделюванням стільки, скільки потрібно, і тоді, коли їм це було потрібно. Студенти не намагалися створити повністю відпрацьовані моделі для функції, яку вони збирались реалізувати – моделі використовувалися переважно для мозкового штурму та для продумування дизайну. Після того, як дизайн був продуманий достатньо для того, щоб дозволити команді перейти до наступного етапу розробки, моделі не змінювались, проте ними продовжували користуватись як артефактом проектування: студенти, як правило, використовували їх або для нагадування про те, що вони робили, або як фоновий «шум», який певним чином допомагав їм працювати в групі [23, с. 18]. Такий підхід активно застосовується експертами з проектування програмного забезпечення – так, М. Петре (Marian Petre) та А. ван дер Хук (André van der Hoek) у порадни-

Табл. 1. Порівняльна характеристика студійного та традиційного навчання (за [23, с. 15]).

Аспект	**Студійний курс**	**Традиційний курс**	**Приклад: студійний курс Університету Ланкастера**
Фізичне середовище	Наявна фізична кімната – студія, відкрита та змінювана з метою створення різноманітних групових, індивідуальних та соціальних просторів	Стандартна лабораторія	Спеціальна лабораторія з цілодобовим доступом, яку обслуговують самі студенти
Управління студією	Правила використання простору не повинні бути обмежувальними	Лабораторія суворо контролюється університетом	Цілодобовий доступ; дозволено їжу / напої; студенти мають права адміністратора
Режими освіти	Педагогічний персонал відіграє скоріше тренерську / менторську роль, ніж викладацьку	Студентам надається перелік документації, що має бути виготовлена	Студентам пропонується виробити стільки документації, скільки їм потрібно – відсутні нормативи щодо того, які діаграми чи позначення використовувати
Поінформованість	Робота розміщується для огляду (як незавершений або кінцевий продукт) – це допомагає студентам побачити роботи один одного	–	Студенти використовують мобільні дошки в студії для демонстрації роботи з проектування, які залишаються на весь час виконання проекту
Критика	Для надання зворотного зв'язку та розвитку ідей використовується постійна критика (офіційна та неофіційна, групова та індивідуальна, взаємна та викладацька)	Надається лише як частина щотижневих зустрічей з керівником проекту	Забезпечується на постійній основі з використанням різноманітних методів: індивідуальних та групових демонстрацій / презентацій, неформального навчання, взаємної критики, критики з боку зовнішніх експертів (наприклад, компаній) та колегіального оцінювання
Культура	Студійна культура має бути соціальною та сприяти поширенню досвіду і підтримці гарної трудової етики	Відсутність виділеної лабораторії означає, що студенти, як правило, зустрічаються лише в заздалегідь обумовлений час	Студенти використовують студію як дім, що веде до випадкових взаємодій та відчуття належності до спільноти
Натхнення	Під час проектування студентів слід заохочувати до креативності у їх проектах та рішеннях	Специфікація проекту визначена викладачем наперед	Студенти пропонують свої власні ідеї проектів під час керованого креативного мозкового штурму

ку [14] використали неформальне моделювання для узагальнення власного досвіду проектування (рис. 2).

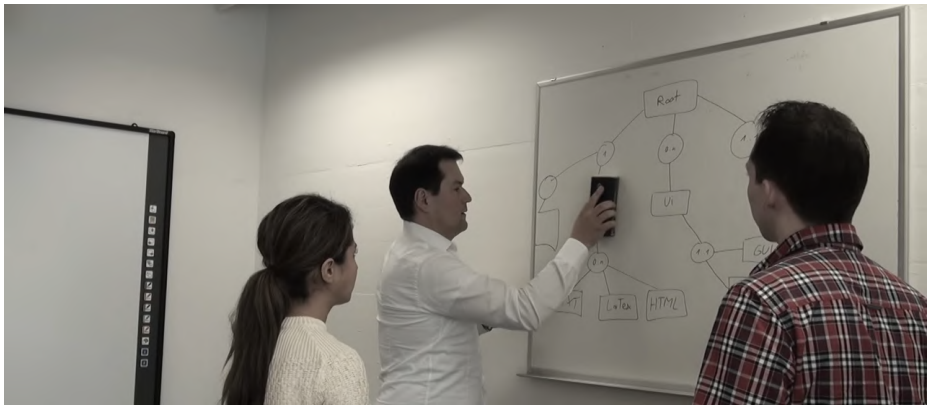

Рис. 1. Неформальне моделювання на дошці.

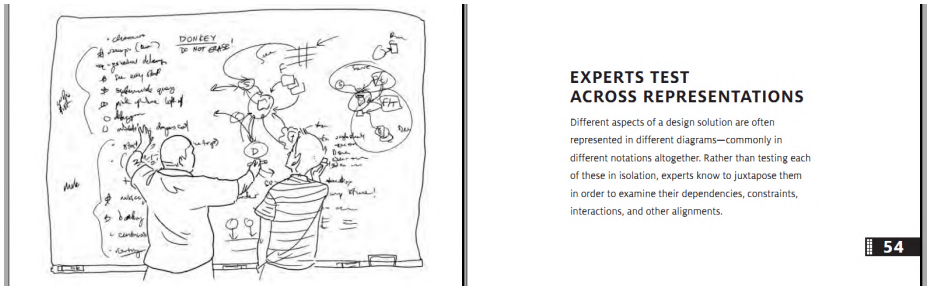

Рис. 2. Фрагмент порадника з проектування М. Петре та А. ван дер Хука [14] у вигляді ескізного проекту.

Поєднання неформального та формального моделювання можливе за допомогою *гнучких засобів моделювання* (рис. 3), які після створення неформального ескізу надають можливість перейти до формального моделювання без зміни засобу [23, с. 17].

Автори студійного курсу [23] вважають, що студентів слід ознайомити із рядом методів проектування, навчити застосувати ці методи на практиці та за допомогою рефлексивної практики заохотити їх дізнатися про позитивні та негативні сторони цих методів: «... можна стверджувати, що найважливіше, чого слід навчити, – це культура самої рефлексивної практики. Якщо студенти навчаться рефлексивним практикам, вони зможуть застосувати ці

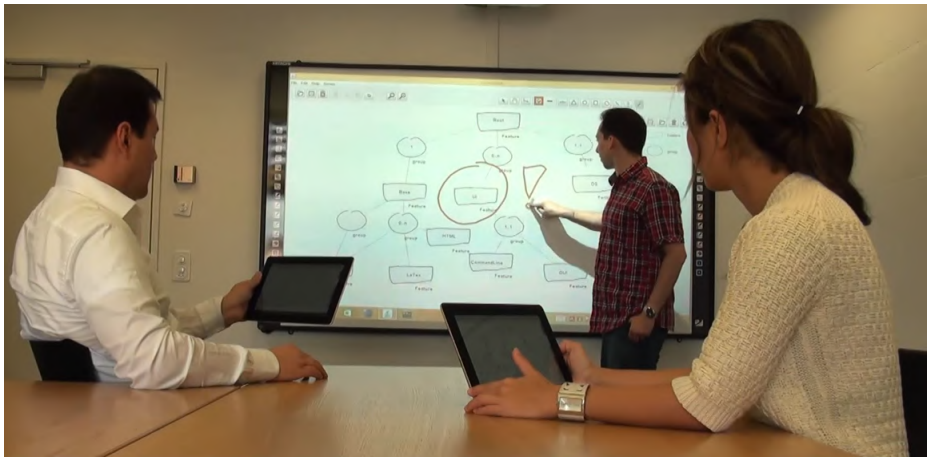

Рис. 3. Моделювання з використанням FlexiSketch Team.

навички до будь-якого нового методу чи підходу, з якими вони зіткнуться у своїй майбутній професійній діяльності» [23, с. 20].

М. Петре та А. ван дер Хук [14, с. 148–158] рекомендують наступні дії з рефлекції при проектуванні програмного забезпечення:

1. Не допускати при обговоренні проекту відхилень на другорядні питання, регулярно перевіряти себе, «де я та що я роблю?».
2. На основі глибокого розуміння фундаментальних концепцій проектування «тримати курс»: пам'ятати, які проектні рішення вже прийняті, чому вони були саме такими та які ще рішення потрібно зробити.
3. Продумування того, що не проектується: визначаючи та розглядаючи обмеження проектування, виявляти спроектоване надмірно або недостатньо.
4. Періодично зупинятись для того, що подивитись на проект у цілому, задаючи собі питання, чи не змінились цілі клієнтів, сприйняття користувачів, сам ринок тощо.
5. Передбачити різні варіанти майбутнього, аналізуючи економічну доцільність для визначення того, куди вкластися – у методи, засоби, ресурси, альтернативи проектування – для того, щоб зберегти зусилля в майбутньому.

У навчанні проектування Ч. Ху рекомендує враховувати наступне [6, с. 69-70]:

1. Технічна компетентність та пізнавальні можливості студентів можуть істотно вплинути на результати навчання проектування. З цієї причини мета викладання проектування програмного забезпечення не в тому, щоб зробити кожного студента дизайнером, а в тому, щоб студенти на практиці відчули, що може знадобитися для створення хорошого проекту, одночасно набуваючи індивідуально досяжних навичок проектування.

2. Для порівняно невеликих завдань проектування керована тестами розробка надає можливість виражати ідеї моделювання безпосередньо у коді, більш ефективно оцінювати компроміси проектування та допускати менше помилок при проектуванні, ніж за використання діаграм.
3. Навчання «проектування у малому» (написанню пов'язаних методів, доцільної інкапсуляції даних, використанню гарних назв методів тощо), ймовірно, є набагато менш складним завданням, ніж вивчення проектування у цілому для дослідження структурну стійкість, яка вимагає проектних компромісів щодо застосування знань про процес проектування. Мінімальні базові здатності до проектування (з відповідним рівнем проектного мислення), які повинні здобути випускники, може включати здатність ефективно розпочинати проектування, декомпозиціювати задачу, виконувати ітерації проектування з обґрунтованими рішеннями щодо компромісів проектування та реалізувати проект у коді.
4. Студенти не можуть ефективно здобути здатності до проектування та навчитись проектного мислення, якщо вони не розв'язують проблеми проектування належної складності.
5. Не зважаючи на відсутність універсальних формул або рецептів проектування, аналіз даних (ідентифікація сутностей та їх відносин) та аналіз процесів (виявлення дій та логічних потоків) є критично важливими для розуміння, яку інформацію використовувати та як вона перетворюється.
6. Правильне документування проекту – своєчасне, а не постфактум – є необхідним навиком, однак, чи повинні студенти використовувати UML або наскільки точно вони використовують мову для документування моделі, не так важливо. Студентів доцільно ознайомити із деякими класичними діаграмами, заохочуючи їх творче використання.
7. Для заключного оцінювання результатів навчання студентів доцільно поєднувати традиційні аудиторні випробування (щоб переконатися, що «вони це знають») з комплексними домашніми задачами проектування, які повинні розв'язуватись окремо або командами з двох осіб із ретельним моніторингом та оцінкою не лише кінцевих продуктів проектування, які виробляють студенти, а й якості виконання проекту, задокументованої у звітах студентів.

3 Висновки та перспективи подальших досліджень

Узагальнення результатів дослідження надало можливість зробити наступні висновки:

1. Проектування програмного забезпечення є різновидом інженерного проектування, у якому об'єднано дві складові – творча та системотехнічна. Формування та розвиток інженерної творчості є технологією з опанування кращих зразків проектів у процесі діяльності з проектування, наближеної до виробничої. Результатом проектування є оновлений та адаптований зразок (типовий проект, або шаблон проектування) чи оригінальний новий дизайн (проект). У зв'язку з цим значний потенціал для

розробки методики навчання проектування програмного забезпечення майбутніх фахівців з ІПЗ наявний у суміжних дослідженнях як з інших галузей інженерії (зокрема, будівельної, механічної та комп'ютерної), так й з мистецтва (зокрема, архітектури, живопису).

2. У навчанні проектування програмного забезпечення на особливу увагу заслуговує підтримка діяльності студентів з проектування, теоретичні основи якого набуті за лекційною формою, у формі практичних занять, студій, курсових проектів тощо із розв'язання реальних задач проектування складного програмного забезпечення. Це висуває додаткову вимогу до викладачів проектування – працювати на виробництві програмного забезпечення або бути тісно пов'язаними із його замовниками.
3. Моделювання програмного забезпечення відіграє ключову роль в його успішному проектуванні. Водночас дотримання вимог використання точних формальних описів моделей у процесі проектування не суттєво впливає на якість проектування: зручність застосування точних формальних та гнучких неформальних моделей є порівняною. У зв'язку з цим доцільним є застосування гнучких методів та засобів моделювання, за яких побудова формальних діаграм UML виконується за неформальним ескізним проектом.
4. Перехід від архітектурного проектування на основі обраного шаблону до неархітектурного (детального) проектування може потребувати адаптації шаблону до непроектних реалій у вигляді інструментів конструювання програмного забезпечення, таких як фреймворку. Тому, незважаючи на те, що проектування програмного забезпечення є можливим без його конструювання, доцільним результатом проектування є принаймні сконструйований прототип програмного забезпечення: будучи частиною життєвого циклу програмного забезпечення, проектування виступає рушійною силою його розвитку – воно постійно виконується аж до виведення програмного продукту з експлуатації.
5. Оцінювання сформованості компетентності з проектування програмного забезпечення передбачає перевірку індивідуальних знань та командних умінь проектування у процесі розв'язання комплексної задачі проектування. Важливими для цього є артефакти проектування – ескізні проекти, формальні моделі, проектна документація, створені прототипи тощо. У зв'язку з цим доцільною формою оцінювання є захист проектів.

Перспективи подальшого розвитку даного дослідження полягають в обґрунтуванні третьої (після інженерії вимог та проектної інженерії) інженерної складової ІПЗ – *конструювання програмного забезпечення.*

References

1. Ali, Z., Bolinger, J., Herold, M., Lynch, T., Ramanathan, J., Ramnath, R.: Teaching object-oriented software design within the context of software frameworks. In: 2011 Frontiers in Education Conference (FIE). pp. S3G–1–S3G–5 (2011). https://doi.org/10.1109/FIE.2011.6142889

2. Carrington, D.: Teaching software design and testing. In: FIE '98. 28th Annual Frontiers in Education Conference. Moving from 'Teacher-Centered' to 'Learner-Centered' Education. Conference Proceedings (Cat. No.98CH36214). vol. 2, pp. 547–550 vol.2 (1998). https://doi.org/10.1109/FIE.1998.738732
3. CC2020 Task Force: Computing Curricula 2020: Paradigms for Global Computing Education. Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3467967
4. Graham, R.M.: Teaching systems programming and software design: Problems and solutions. SIGCSE Bull. **2**(3), 56–60 (nov 1970). https://doi.org/10.1145/873641.873652
5. Hamilton, Jr., J.A., Murtagh, J.L.: Teaching a real world software design approach within an academic environment. In: 2000 Annual Conference. p. 5.577.1–5.577.8. No. 10.18260/1-2–8739, ASEE Conferences, St. Louis, Missouri (June 2000), https://peer.asee.org/8739
6. Hu, C.: The nature of software design and its teaching: An exposition. ACM Inroads **4**(2), 62–72 (jun 2013). https://doi.org/10.1145/2465085.2465103
7. Jarzabek, S.: Teaching advanced software design in team-based project course. In: 2013 26th International Conference on Software Engineering Education and Training (CSEE T). pp. 31–40 (2013). https://doi.org/10.1109/CSEET.2013.6595234
8. Jia, Y., Tao, Y.: Teaching software design using a case study on model transformation. In: 2009 Sixth International Conference on Information Technology: New Generations. pp. 702–706 (2009). https://doi.org/10.1109/ITNG.2009.114
9. Joint Task Force on Computing Curricula, IEEE Computer Society, Association for Computing Machinery: Software Engineering 2014: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering. A Volume of the Computing Curricula Series (2015), https://www.acm.org/binaries/content/assets/education/se2014.pdf
10. Lamm, E.: Booch's Ada vs. Liskov's Java: Two approaches to teaching software design. In: Rosen, J.P., Strohmeier, A. (eds.) Reliable Software Technologies — Ada-Europe 2003. pp. 102–112. Springer Berlin Heidelberg, Berlin, Heidelberg (2003). https://doi.org/10.1007/3-540-44947-7_7
11. Ministerstvo osvity i nauky Ukrainy: Pro zatverdzhennia standartu vyshchoi osvity za spetsialnistiu 121 "Inzheneriia prohramnoho zabezpechennia" dlia pershoho (bakalavrskoho) rivnia vyshchoi osvity (On approval of the standard of higher education in the specialty 121 "Software Engineering" for the first (bachelor's) level of higher education) (2018), https://mon.gov.ua/storage/app/media/vishcha-osvita/zatverdzeni%20standarty/12/21/121-inzheneriya-programnogo-zabezpechennya-bakalavr.pdf
12. van Niekerk, J., Futcher, L.: The use of software design patterns to teach secure software design: An integrated approach. In: Bishop, M., Miloslavskaya, N., Theocharidou, M. (eds.) Information Security Education Across the Curriculum. pp. 75–83. Springer International Publishing, Cham (2015). https://doi.org/10.1007/978-3-319-18500-2_7
13. Pelevin, V.N.: Formirovanie professionalnoi kompetentnosti budushchikh bakalavrov po napravleniiu "Informatcionnye sistemy i tekhnologii" (Formation of professional competence of future bachelors in the direction of "Information systems and technologies"). Dissertation for degree of candidate of pedagogical sciences: 13.00.08 – theory and methods of vocational education, Ural State Technical University – UPI named after the first President of Russia B. N. Yeltsin (2010), https://elar.rsvpu.ru/handle/123456789/987

14. Petre, M., van der Hoek, A.: Software Design Decoded: 66 Ways Experts Think. The MIT Press, Cambridge (2016)
15. Pierce, K., Deneen, L., Shute, G.: Teaching software design in the freshman year. In: Tomayko, J.E. (ed.) Software Engineering Education. pp. 219–231. Springer Berlin Heidelberg, Berlin, Heidelberg (1991). https://doi.org/10.1007/BFb0024294
16. Semerikov, S., Striuk, A., Striuk, L., Striuk, M., Shalatska, H.: Sustainability in software engineering education: a case of general professional competencies. E3S Web of Conferences **166**, 10036 (2020). https://doi.org/10.1051/e3sconf/202016610036
17. Striuk, A.M.: Software engineering: First 50 years of formation and development. CEUR Workshop Proceedings **2292**, 11–36 (2018)
18. Striuk, A.M., Semerikov, S.O.: The dawn of software engineering education. CEUR Workshop Proceedings **2546**, 35–57 (2019), http://ceur-ws.org/Vol-2546/paper02.pdf
19. Striuk, A.M., Semerikov, S.O., Shalatska, H.M., Holiver, V.P.: Software requirements engineering training: problematic questions. CEUR Workshop Proceedings pp. 3–11 (2022)
20. Tamburri, D.A., Razavian, M., Lago, P.: Teaching software design with social engagement. In: 2013 26th International Conference on Software Engineering Education and Training (CSEE T). pp. 61–69 (2013). https://doi.org/10.1109/CSEET.2013.6595237
21. Warren, I.: Teaching patterns and software design. In: Proceedings of the 7th Australasian Conference on Computing Education - Volume 42. p. 39–49. ACE '05, Australian Computer Society, Inc., AUS (2005). https://doi.org/10.5555/1082424.1082430
22. Weiss, D.M.: Teaching a software design methodology. IEEE Transactions on Software Engineering **SE-13**(11), 1156–1163 (1987). https://doi.org/10.1109/TSE.1987.232864
23. Whittle, J., Bull, C.N., Lee, J., Kotonya, G.: Teaching in a software design studio: Implications for modeling education. CEUR Workshop Proceedings **1346**, 12–21 (2014), http://ceur-ws.org/Vol-1346/edusymp2014_paper_1.pdf
24. Williams, C., Kurkovsky, S.: Raspberry Pi creativity: A student-driven approach to teaching software design patterns. In: 2017 IEEE Frontiers in Education Conference (FIE). pp. 1–9 (2017). https://doi.org/10.1109/FIE.2017.8190735