МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ ЛІНГВІСТИЧНИЙ УНІВЕРСИТЕТ

Кафедра теорії і практики перекладу з англійської мови

Кваліфікаційна робота магістра з перекладознавства

на тему: «Стратегії передачі професіоналізмів у контексті програмування ІТ дискурсу»

Студентки групи МПа 06-21
факультету германської філології і перекладу
освітньо-професійної програми
Перекладознавство: професійно-орієнтований переклад
(англійська мова і друга іноземна мова)
за спеціальністю 035 Філологія
Оберемок Вікторії Олександрівни

Допущена до захисту
«____»_____ 2022 року

Завідувач кафедри теорії і практики перекладу з англійської мови
_____ доц. Мелько Х.Б.
(підпис) (ПІБ)

Науковий керівник:
кандидат філологічних наук,
доцент Подсєвак К. С.

Національна шкала _____
Кількість балів: _____
Оцінка:ЄКТС _____

Київ – 2022

MINISTRY OF EDUCATION AND SCIENCE OF UKRAINE

KYIV NATIONAL LINGUISTIC UNIVERSITY

Department of Theory and Practice of Translation from the English Language

Master Degree Thesis in Translation Studies

under the title: "Strategies for conveying professionalisms when translating programming texts in IT discourse"

Group MPa 06-21
Faculty of German Philology and Translation
Educational Programme Translation Studies: Specialized Translation (English and Second Foreign Language)
Majoring 035 Philology
Viktoriia O. Oberemok

Research supervisor:
K. S. Podsievak
Candidate of Philology,
Associate Professor

Kyiv – 2022

Київський національний лінгвістичний університет
Кафедра теорії і практики перекладу з англійської мови

ЗАВДАННЯ
на кваліфікаційну роботу магістра з перекладознавства

студентки ІІ курсу групи МПа 06-21 факультету германської філології і перекладу КНЛУ
Оберемок Вікторії Олександрівни
(ПІБ студента)
спеціальності 035 Філологія, **спеціалізації** 035.041 Германські мови і літератури (переклад включно), перша – англійська, **освітньо-професійної програми** Перекладознавство: професійно-орієнтований переклад (англійська мова і друга іноземна мова)
Тема роботи «Стратегії передачі професіоналізмів у контексті програмування ІТ дискурсу»
Науковий керівник_ кандидат філологічних наук, доцент Подсєвак К. С._____
Дата видачі завдання__"10" вересня 2021 р._____

Графік виконання кваліфікаційної роботи магістра з перекладознавства

№ п/п	Найменування частин і план кваліфікаційної роботи	Графік виконання	Підписи студента і керівника
1.	Аналіз наукових першоджерел і складання бібліографії	Жовтень 2021 р.	
2.	Написання теоретичної частини кваліфікаційної роботи (розділ 1)	Листопад 2021 р.	
3.	Добір мовного матеріалу тексту і складання Додатку (100 англійськомовних речень та їх переклад)	Грудень 2021 р.	
4.	Аналіз мовного матеріалу тексту, який досліджується, і написання аналітичної частини кваліфікаційної роботи (розділ 2)	Березень 2022 р.	
5.	Проведення перекладацького аналізу досліджуваного мовного явища і написання практичної частини кваліфікаційної роботи (розділ 3)	Травень 2022 р.	
6.	Написання вступу і висновків дослідження, подання завершеної кваліфікаційної роботи науковому керівнику для попереднього перегляду	Вересень 2022 р.	
7.	Попередній захист кваліфікаційної роботи і подання завершеної кваліфікаційної роботи на кафедру	07 жовтня 2022 р.	
8.	Оформлення документації (відгуки) і підготовка презентації до захисту кваліфікаційної роботи	Жовтень 2022 р.	
9.	Захист кваліфікаційної роботи магістра з перекладознавства	Грудень 2022 р.	

Науковий керівник _____ (підпис)
Студент _____ (підпис)

ВІДГУК НАУКОВОГО КЕРІВНИКА
НА КВАЛІФІКАЦІЙНУ РОБОТУ МАГІСТРА З ПЕРЕКЛАДОЗНАВСТВА

студентки ІІ курсу групи МПа 06-21 факультету германської філології і перекладу спеціальності 035 Філологія, **спеціалізації** 035.041 Германські мови і літератури (переклад включно), перша – англійська, **освітньо-професійної програми** Перекладознавство: професійно-орієнтований переклад (англійська мова і друга іноземна мова)

Оберемок Вікторії Олександрівни

(ПІБ студента)

за темою «Стратегії передачі професіоналізмів у контексті програмування ІТ дискурсу»

Відповідність кваліфікаційної роботи нормативним вимогам (необхідне позначити ∨ або +)		
1.	Наявність основних структурних компонентів	_____ усі компоненти присутні , _____ один компонент відсутній _____ декілька компонентів відсутні
2.	Відповідність оформлення, посилань і списку використаних джерел нормативним вимогам	_____ повна відповідність _____ незначні помилки в оформленні _____ оформлення неправильне
3.	Відповідність побудови вступу нормативним вимогам	_____ повна відповідність _____ відповідність неповна _____ не відповідає вимогам
4.	Відповідність огляду наукової літератури нормативним вимогам	_____ повна відповідність _____ відповідність неповна _____ не відповідає вимогам
5.	Відповідність аналітичної частини дослідження заявленій меті та завданням	_____ повна відповідність _____ відповідність неповна _____ не відповідає вимогам
6.	Відповідність практичної частини дослідження нормативним вимогам	_____ повна відповідність _____ відповідність неповна _____ не відповідає вимогам
7.	Відповідність висновків результатам теоретичної та практичної складових дослідження	_____ повна відповідність _____ відповідність неповна _____ не відповідає вимогам

Особиста думка керівника_____
_____
_____
_____
_____
_____
_____
_____

Кваліфікаційна робота_____**може бути (не може бути)**
(ПІБ студента)
рекомендована до захисту

_____ **(**_____**)**
(підпис керівника) (ПІБ керівника)

”____” _____ 2022 рік

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ МАГІСТРА
З ПЕРЕКЛАДОЗНАВСТВА

студентки ІІ курсу групи МПа 06-21 факультету германської філології і перекладу спеціальності 035 Філологія, **спеціалізації** 035.041 Германські мови і літератури (переклад включно), перша – англійська, **освітньо-професійної програми** Перекладознавство: професійно-орієнтований переклад (англійська мова і друга іноземна мова)

Оберемок Вікторії Олександрівни

(ПІБ студента)

за темою «Стратегії передачі професіоналізмів у контексті програмування ІТ дискурсу»

	Критерії	**Оцінка в балах**
1.	Наявність основних компонентів структури роботи — ***загалом 10 балів*** (усі компоненти присутні – **10**, один компонент відсутній – **5**, декілька компонентів відсутні – **0**)	
2.	Відповідність оформлення роботи, посилань і списку використаних джерел нормативним вимогам до кваліфікаційної роботи — ***загалом 10 балів*** (повна відповідність – **10,** поодинокі огріхи у форматуванні – **8**, незначні помилки в оформленні – **6**, значні помилки в оформленні – **4**, оформлення переважно не відповідає вимогам **– 0**)	
3.	Відповідність побудови вступу нормативним вимогам — ***загалом 10 балів*** (повна відповідність – **10**, поодинокі огріхи стилістичного характеру – **8**, несуттєві помилки у формулюваннях – **6**, суттєві помилки у формулюваннях – **4**, не відповідає вимогам за структурою і змістом – **0**)	
4.	Відповідність огляду наукової літератури нормативним вимогам — ***загалом 10 балів*** (повна відповідність – **10**, несуттєві помилки у формулюваннях – **8**, недостатня кількість проаналізованих іноземних джерел (мін. 30%) – **6**, відсутній критичний аналіз наукових праць – **4**, не відповідає вимогам за структурою і змістом – **0**)	
5.	Відповідність аналітичної частини дослідження заявленій меті та завданням — ***загалом 10 балів*** (повна відповідність – **10**, несуттєві огріхи стилістичного характеру – **8**, несуттєві помилки при аналізі фактичного матеріалу –**6**, суттєві помилки при аналізі фактичного матеріалу – **4**, відсутність власного аналізу фактичного матеріалу (100 речень) – **0**)	
6.	Відповідність практичної частини дослідження нормативним вимогам — ***загалом 10 балів*** (повна відповідність – **10**, несуттєві огріхи стилістичного характеру – **8**, несуттєві помилки при перекладі фактичного матеріалу – **6**, суттєві помилки при перекладі й аналізі фактичного матеріалу – **4**, відсутність перекладацького аналізу фактичного матеріалу (100 речень) – **0**)	
7.	Відповідність висновків результатам теоретичної та практичної складових дослідження — ***загалом 10 балів*** (повна відповідність – **10**, несуттєві огріхи стилістичного характеру – **8**, неповне висвітлення результатів дослідження – **6**, часткове висвітлення результатів дослідження – **4**, не відповідає результатам дослідження – **0**)	

Усього набрано балів: _____

_____ _____

(ПІБ рецензента) (підпис рецензента)

”____” _____ 2022 р

ЗМІСТ

ВСТУП

Кваліфікаційну роботу магістра присвячено вивченню особливостей передачі професіоналізмів у контексті програмування ІТ дискурсу на прикладі матеріалів з програмування мовою Java.

Виклики інформатизації й глобалізації, розширення міжкультурних та економічних зв'язків на світовому рівні визначають потребу у міжнародному співробітництві сфері інформаційних технологій, а особливо – у контексті програмування, оскільки ця діяльність є визначальною для багатьох підприємств і організацій. Таким чином, підвищується необхідність у комунікації між спеціалістами в цій сфері, яка ускладняється для українськомовних програмістів тим фактом, що більшість професійної літератури з програмування націлена на англійськомовного читача. Зокрема, складність при такій комунікації створюють професіоналізми, які спеціаліст повинен розуміти чітко і ясно, від чого залежить загалом ефективність його професійної діяльності.

Проблемам вивчення професфоналізмів висвітлена в роботах таких вітчизняних а зарубіжних науковців, як А. Л. Арцишевська, І. А. Балас, О. Г. Берестенко, Ю. В. Бугаєвська, С. З. Булик-Верхола, С. А. Вискушенко, Л. Гоффман, Ю. І. Дрозябко, М. Г. Зубков, О. С. Кацан, Т. Р. Кияк, І. І. Ковтун, З. І. Комарова, Л. Ю. Корольова, Н. П. Кулічова, О. Павлова, О. І. Павлюк, Т. Роелке, О. В. Слюсаренко, Л. Томіленко, С. В. Харицька, С. Д. Шелов та ін. Особливості перекладу професійної лексики розглядають у своїх наукових працях Т. Р. Ананко, М. В. Бережна, Я. І. Каламбет, В. І. Карабан, Т. Р. Кияк, А. О. Колесник, С. В. Міхєєва, А. Л. Міщенко, І. В. Онушканич, Л. Черноватий, С. В. Шепітько. Однак попередній аналіз літератури за напрямом демонструє недостатню дослідженість професіоналізмів у контексті програмування ІТ дискурсу як з лінгвістичної, так і з перекладознавчої точки зору.

Тож **актуальність дослідження** зумовлена як недостатньою вивченістю проблеми, так і інтересом лінгвістів і перекладачів до проблем мовних одиниць дискурсу професійного спілкування та необхідністю визначити шляхи перекладу професійної лексики в контексті програмування ІТ дискурсу. До того ж, актуальність дослідження також визначається все більш стрімким зростанням міжнародної комунікації у сфері програмування, що вимагає вирішення практичних проблем з метою покращення якості такої комунікації.

Мета дослідження – проаналізувати стратегії та конкретні засоби передачі при перекладі українською мовою англійськомовних професіоналізмів у контексті програмування ІТ дискурсу.

Досягнення мети дослідження передбачає виконання таких **завдань**:

1) розглянути професіоналізм як об’єкт лінгвістичного дослідження;
2) дослідити професіоналізм як перекладознавчу проблему;
3) визначити особливості репрезентації діяльності з програмування у дискурсі інформаційних технологій;
4) проаналізувати семантичні параметри професіоналізмів у контексті програмування ІТ дискурсу;
5) виявити особливості творення професіоналізмів у контексті програмування ІТ дискурсу;
6) висвітлити лексичні перекладацькі трансформації при перекладі професіоналізмів у контексті програмування ІТ дискурсу;
7) проаналізувати лексико-семантичні перекладацькі трансформації як засіб передачі професіоналізмів при перекладі у контексті програмування ІТ дискурсу;
8) охарактеризувати граматичні перекладацькі трансформації та їх використання при перекладі професіоналізмів у контексті програмування ІТ дискурсу.

Об’єктом дослідження постають професіоналізми в контексті програмування ІТ дискурсу в англомовних текстах та їх перекладах.

Предмет дослідження – структурно-семантичні особливості та перекладацькі трансформації, що використовуються при відтворенні в україномовному перекладі англійськомовних професіоналізмів у контексті програмування ІТ дискурсу.

Матеріалом дослідження слугують 100 текстових фрагментів, вилучених із тексту, присвяченому мові програмування Java на офіційному сайті її розробника “Sun Microsystems” як підрозділу компанії “Oracle” засобами суцільної вибірки. Загальний обсяг проаналізованих професіоналізмів становить 100 одиниць; у ході дослідження проаналізовано застосування 111 перекладацьких трансформацій.

Для розв’язання поставлених завдань у роботі застосовано комплексну методику дослідження. Основними залученими **методами** стали метод суцільної вибірки, що уможливив добір ілюстративного матеріалу дослідження; методи семантичного та компонентного аналізу дозволили провести комплексний аналіз професіональної лексики з точки зору лінгвістики; трансформаційний аналіз використовувався з метою виявлення шляхів відтворення професіоналізмів при перекладі; узагальнення інформації проводилося з використанням методів кількісного аналізу.

Наукова новизна проведеного дослідження полягає в тому, що в роботі розроблено тематичну класифікацію професіоналізмів у контексті програмування ІТ дискурсу та визначено його характерні риси. Окрім того, в роботі здійснено структурний аналіз професіоналізмів у контексті програмування ІТ дискурсу, а саме, визначено структуру цих мовних одиниць відповідно до кількості компонентів та в подальшому розподілено їх за групами відповідно до способів творення. До того ж, у роботі визначено основні трансформації, що використовуються при відтворенні в україномовному перекладі англійськомовних професіоналізмів у контексті програмування ІТ дискурсу.

Практична цінність роботи визначається тим, що здійснене в ній дослідження структурно-семантичних особливостей професіоналізмів у контексті програмування ІТ дискурсу та особливостей їх відтворення при перекладі становить внесок до теорії зіставного мовознавства, а також в теорію міжмовних контактів, мови для спеціальних цілей та дискурсологію.

Результати проведеного дослідження можуть бути впроваджені при викладанні курсів лексикології англійської мови та галузевого перекладу. Отримана інформація також може бути використана в лексикографії з метою кодифікації та уніфікації професійної лексики сфери програмування та при створенні тлумачних або двомовних словників професіоналізмів.

Структура й обсяг роботи. Кваліфікаційна робота складається зі вступу, трьох розділів з висновками до кожного з них, загальних висновків, списків використаних джерел, довідкової літератури, джерел ілюстративного матеріалу, двох додатків та резюме.

РОЗДІЛ 1
ТЕОРЕТИЧНІ ПЕРЕДУМОВИ ВІДТВОРЕННЯ ПРИ ПЕРЕКЛАДІ ПРОФЕСІОНАЛІЗМІВ У КОНТЕКСТІ ПРОГРАММУВАННЯ ІТ ДИСКУРСУ

1.1 Професіоналізм як об'єкт лінгвістичного дослідження

Комунікація – це невід'ємна умова діяльності людини. Це складний процес установлення й розвитку контактів між людьми, що породжується потребами в спільній діяльності і включає обмін інформацією, сприймання й розуміння між людьми, вироблення єдиної стратегії взаємодії [6: 15]. Правильно організована комунікація забезпечує ефективний обмін інформацією, дає змогу глибше пізнати співрозмовника, спрогнозувати особливості подальшої ділової взаємодії з партнером [61: 150], а професійна комунікація дозволяє точно і ємно передати знання та досвід. Засобом професійної комунікації постає мова для спеціальних цілей.

Термін «мова для спеціальних цілей» уживається в лінгвістичних працях на позначення всіх форм вербальної і невербальної фахової комунікації в межах науково-технічної галузі. Крім того, різні дефініції терміну «мова для спеціальних цілей» наголошують на релевантних властивостях фахових мов з огляду на коло наукових інтересів дослідника [85: 448].

За Ж. В. Красножан [37], мова для спеціальних цілей – це сукупність усіх лексичних номінацій комунікативно-професійної сфери. Вона є основою будь-якого професійного мовлення. Мова для спеціальних цілей продуктивно використовують у нефахових текстах: художніх, публіцистичних, розмовних, збагачуючи та урізноманітнюючи стилістичні можливості мови. Спеціальні лексеми, що потрапили в художній чи публіцистичний текст, вживаючись у переносному значенні, виконують цілком іншу функцію [37: 10].

Найбільш розгорнуте та релевантне висвітлення поняття «мова для спеціальних цілей» простежується в німецького лінгвіста Л. Гофмана [74], який висуває тезу, відповідно до якої **мова для спеціальних цілей** – це сукупність усіх мовних засобів, які застосовуються в певній обмеженій фахом галузі комунікації для забезпечення взаєморозуміння між зайнятими в цій галузі фахівцями [74: 53]. Під сукупністю мовних засобів автор розуміє не тільки фонетичні, морфологічні і лексичні елементи та правила синтаксису, але й їх взаємодію. Автор розрізняє: (а) мовні засоби, які наявні у всіх субмовах, (б) мовні засоби, які наявні у всіх фахових мовах, та (в) мовні засоби, які наявні тільки в одній фаховій мові [74: 53].

Відповідно до найбільш загальної **класифікації**, мови для спеціальних цілей поділяють на наукові (мови астрономії, генетики) та ті, що наближаються до практичних потреб сучасного суспільства. Крім того, всі мови для спеціальних цілей можна класифікувати «горизонтально» і «вертикально». Горизонтальний розподіл включає в себе розмежування мов для спеціальних цілей за предметними галузями. Таким чином, у межах мов для спеціальних цілей прийнято розрізняти мови соціальних наук і технічних галузей, які мають відмінності не лише на рівні терміносистеми, але й на рівні організації тексту й мовної структури. Вертикальний розподіл передбачає диференціацію мов для спеціальних цілей за рівнем абстрактності, типом мови з погляду природності-штучності, складом учасників комунікації, сферою функціонування. Важливо зазначити, що хоча вертикальні класифікації вважаються допоміжними, їх важливість не підлягає запереченню [12: 142].

Особливість мов для спеціальних цілей, як справедливо зазначає Т. Р. Кияк [24], полягає в наявності спеціального орієнтованого на потреби певного фаху набору лексичних одиниць, які мають усе ж плавні та гнучкі зв’язки із загальновживаною лексикою, яка теж наявна в мові для спеціальних цілей. З іншого боку, остання відрізняється специфічною частотою вживання певних граматичних, синтаксичних, стилістичних засобів [24: 22]. Згідно зі

словником лінгвістичних термінів, **спеціальна лексика** – це сукупність слів чи словосполучень, що позначають поняття спеціальної галузі знання або діяльності (СЛТ: 359).

На стилістичному рівні мова для спеціальних цілей включає три **підкласи лексичних одиниць**:

1) спеціальні терміни (власне терміни) – лексичні одиниці терміносистем, що входять до лексики тієї чи іншої мови для спеціальних цілей, а також загальнонаукові, загально технічні та міжгалузеві терміни, спільні для більшості мов для спеціальних цілей;

2) терміноїди – широкий клас лексичних одиниць, що не відповідають вимогам до терміну з різних причин;

3) псевдотерміни – лексичні одиниці мови для спеціальних цілей, що позначають теорії, які визнаються хибними [33: 139].

На семантичному рівні усю лексику мови для спеціальних цілей можна поділити на:

1) терміни даної галузі, які мають свою власну дефініцію;

2) міжгалузеві загальнонаукові термінологічні одиниці (у тому числі терміни суміжних наук);

3) професіоналізми, до яких також можна віднести номенклатуру (особливі типи термінів, які співвідносяться з одиничними поняттями і актуалізують предметні зв’язки);

4) професійні жаргонізми [25: 140].

Усі спеціальні лексеми виконують дві основні **функції**:

1) функцію позначення, найменування вузькопрофесійних спеціальних понять і систем понять (предметів, ознак, дій, процесів);

2) функцію особливого найменування загальновідомих понять, що надає цьому поняттю підвищеної виразності, експресивності, емоційності [3: 5].

Спеціальну лексику часто протиставляють загальновживаній лексиці на основі того, що вона відрізняється, по-перше, своєю змістовою пов’язаністю з

об’єктами певної галузі, по-друге, тим, що в межах фахового комунікація має високу частотність, а з погляду словникового складу загалом лише незначною мірою входить до сфери достатньо частотної. Однак більшість науковців вважає, що чітко протиставити спеціальну лексику загальновживаній (нетермінологічній) неможливо. Між ними лежить широка смуга, у якій терміни існують ніби в постійному коливанні між ідеальними вимогами (однозначність, нейтральність, відсутність синонімів) і реальними законами живої та динамічної лексичної системи [58: 184]. Саме в цій «перехідній» смузі відбувається активна взаємодія між термінами й не термінами, тому користувачі у власній мовленнєвій діяльності вживають їх нарівні. Внаслідок таких взаємодій відбуваються системні переходи лексики з одних царин в інші: термінів у царину загальної лексики (детермінологізація) і навпаки – поповнення термінів за рахунок ресурсів загальної лексики (термінологізація) [60: 17].

До складу спеціальної лексики також належать професіоналізми й професійні сленгізми. У більшості випадків професіоналізми вживаються в усному неофіційному мовленні [39: 269]. Певний час уважали, що спеціальна лексика – це лексика, що розвинулася й досягла розвитку в період ремісничого виробництва [58: 183].

За М. Г. Зубковим [19], **професіоналізмами** є слова й мовленнєві звороти, характерні для мови людей певних професій. Оскільки професіоналізми вживають на позначення спеціальних понять лише у сфері тієї чи іншої професії, ремесла, промислу, вони не завжди відповідають нормам літературної мови [19: 55]

О. Палюк [48] дає таку дефініцію: професіоналізми – мовні одиниці, що загалом відповідають системі української мови і реалізовуються в усній мові професіоналів під час неофіційного комунікація [48: 85].

Найбільш повним є визначення В. М. Пивоварова [51], за яким професіоналізми – це напівофіційні слова, сферою уживання яких є, як

правило, усне побутово-професійне мовлення. Тож під професіоналізмами розуміють слова або звороти, властиві мовленню людей певної професії, здебільшого вони застосовуються в усному неофіційному мовленні людей певного фаху. Оскільки вони вживаються на позначення спеціальних понять лише у сфері тієї чи іншої професії, ремесла, промислу, то не завжди відповідають нормам літературної мови і можуть мати навіть жаргонний характер [51: 169].

Професіоналізми – поняття, що детально диференціюють ті предмети, дії, якості, що безпосередньо пов'язані зі сферою діяльності відповідної професії [19: 55], наприклад: *framework* ‘програмний продукт, система чи платформа, основна мета якої – полегшення, спрощення, створення і підтримка різного роду проектів, програм, послуг і сайтів’; *deploy* ‘розгортання (перенесення) програмного забезпечення (виконуваного коду) на сервер або пристрій, де воно буде працювати’; *hackathon* ‘захід, форум, на якому збираються фахівці з різних галузей розробки програмного забезпечення (найчастіше програмісти, але можуть бути і дизайнери, тестери) для розробки певного проекту, додатку чи програми’. Напівофіційна мова, що не закріпилася у словниках, проте активно використовується у професійному середовищі, виникає як необхідність назвати той чи інших технологічний процес, операцію, дію [23: 183].

Потрібно зазначити, що професіоналізми виконують не лише інформативну, а й емотивну, оцінювальну **функції**, оскільки лексичне значення професіоналізмів на відміну від термінів складається з когнітивного й прагматичного компонентів. Через конотативну забарвленість професіоналізмів виявляється суб’єктивне експресивне ставлення комуніканта до об’єкта комунікація. Завдяки прагматичній спрямованості професіоналізмів їхнє вживання також зумовлено наміром комуніканта реалізувати власні імпліцитні цілі комунікація, зокрема встановити дружні

стосунки зі співрозмовником [70: 124], що досягається через зрівнювання соціального та професійного рівнів комунікантів.

На проблему співвідношення термінології та професійної лексики існують, принаймні, три погляди. Перший з них ототожнює ці два поняття; другий – розмежовує професійну лексику й термінологію деякою історико-тематичною рисою; третій погляд, констатуючи наявність значної спільної частини цих лексичних шарів, може пояснити й існування частин, які не збігаються [47: 51–52]. Останній з них виділяє такі особливості, які допомагають відрізнити професіоналізми від термінів:

1) професіоналізми належать до ненормативної спеціальної лексики на відміну від термінів, які є нормативною частиною спеціальної лексики наукової мови;

2) професіоналізми зрідка подають у загальних та спеціальних словниках й існують переважно у сфері функціонування, на відміну від термінів, які фіксуються словниками і функціонують одночасно у двох сферах (фіксації та функціонування);

3) домінантною сферою функціювання термінів є письмове мовлення, а професіоналізми використовують переважно в розмовному мовленні [57: 20];

4) професіоналізми мають дещо ширшу сферу спеціальної діяльності; терміни ж можуть бути відомі навіть людям, не пов’язаним із окресленою професійною сферою;

5) професіоналізми виникають в умовах професійного комунікація як вторинні форми вираження і функціонують найчастіше як професійно-розмовні дублети офіційних термінів;

6) у професіоналізмах певної галузі системні зв’язки виражено меншою мірою, ніж у термінах [31: 18];

7) професіоналізми характеризуються прагненням до виразності, образності, експресії, на відміну від термінів, які позбавлені конотації;

8) у професіоналізмах спостерігається менша, порівняно з термінами, спеціалізація словотвірних засобів;

9) у сфері професіоналізмів помітна тенденція до скорочення спеціальних виразів, які застосовують в професійному мовленні дуже часто;

10) професіоналізми належать до периферії відповідної термінологічної системи, тимчасом як терміни належать до її центру [31: 18; 47: 51; 57: 21].

Утім, навіть ті, хто вважає доцільною диференціацію термінів та професіоналізмів, наголошують на тому, що чітко окресленої межі між ними немає, вона мобільна й умовна, оскільки між указаними двома шарами спеціальної лексики існує постійний взаємообмін [58: 183].

Професіоналізми виникають у двох випадках: або коли певна галузь не має розвиненої термінології, або як неофіційні замінники термінів, які вживають у розмовній мові. Серед професіоналізмів переважають слова загальнонародної мови, ужиті в незвичайному для них значенні або формі [35: 28–29]. Але, їх влучно використовують у літературі з метою створення професійного колориту, відтворення життєдіяльності певного професійного середовища у своїх творах [21: 145]. Правда, не всі професіоналізми можна віднести до лексики «високого» стилю.

У межах одного колективу, підприємства, відомства може народжуватися безліч нових професіоналізмів, та їх **творення** здійснюється переважно шляхом:

1) семантичний спосіб, що полягає в переосмисленні загальновживаних або спеціальних слів при введенні їх до нової професійної мовної сфери, напр.: *default* ‘за замовчуванням’; *device* ‘пристрій’; *domain* ‘обмежена підмножина значень певного типу’;

2) словоскладання: *upgrade* ‘оновлення’; *software* ‘програмне забезпечення’; *multitasking* ‘багатозадачність’;

3) суфіксація: *bodyshopper* ‘людина, яка займається найманням програмістів у країнах, що розвиваються, для американських комп’ютерних фірм’; *user* ‘користувач’;

4) префіксація: *uninstall* ‘видалити’ [53: 74].

На думку І. Ковтун [29], професіоналізми розпізнають за тими ж ознаками, що й терміноодиниці, але з підкресленням їхньої невідповідності тим основним критеріям, що їх висувають до термінів [29: 65].

Отже, мова для спеціальних цілей – це сукупність усіх мовних засобів, які застосовуються в певній обмежуваній фахом галузі комунікації для забезпечення взаєморозуміння між зайнятими в цій галузі фахівцями. Вона включає три підкласи: спеціальні терміни, терміноїди, псевдотерміни, до цієї мови належать і професіоналізми, професійні жаргонізми. Професіоналізми – це напівофіційні слова, сферою уживання яких є, як правило, усне побутово-професійне мовлення. Професіоналізми виникають у двох випадках: коли певна галузь не має розвиненої термінології, або ж як неофіційні замінники термінів, які вживають у розмовній мові.

1.2 Професіоналізм як перекладознавча проблема

В. Н. Комісcаров [32] виділяє п’ять основних нормативних вимог до перекладу:

1) норма еквівалентності перекладу (комунікативна рівноцінність текстів оригіналу та перекладу);

2) жанрово-стилістична норма перекладу (вимога відповідності перекладу домінантної функції, типу та стилістичним особливостям тексту, до якому належить переклад);

3) норма перекладацької мови (взаємодія правил норми та узусу мови);

4) конвенційна норма перекладу (вимога максимальної близькості перекладу до оригіналу);

5) прагматична норма перекладу (вимога забезпечення прагматичної цінності перекладу) [32: 227–228].

Головною метою будь-якого перекладу – є досягнення адекватності. **Адекватний переклад** – відтворення єдності змісту і форми оригіналу засобами іншої мови. Адекватний переклад враховує і змістову, і прагматичну еквівалентність, не порушуючи при цьому ніяких норм, є точним і без усіляких неприпустимих перекручень. Оскільки адекватний переклад має оціночний характер, то варто розглянути таке поняття як повноцінність перекладу [13: 342].

Повноцінність перекладу полягає в передачі для вихідного тексту співвідношення змісту і форми, шляхом відтворення особливостей останньої (якщо це можливо за мовними умовами), або створення функціональних відповідників цих особливостей. Повноцінний переклад передбачає рівновагу між цілим та окремим і визначає специфіку твору (змісту і форми). Дослівна передача окремо взятих елементів не означає ще повноцінної передачі цілого, оскільки останнє не є звичайною сумою цих елементів, а являє собою певну систему [13: 342].

Професіоналізми вважаються складною категорією з точки зору перекладу, оскільки сфера їх вживання досить обмежена і вони не зафіксовані в словниках [68: 176].

Як правило, вони мають чітко визначену сферу використання (географічну чи професійну) і, або взагалі не мають відповідника-професіоналізму у мові перекладу, або мають переклад, заснований на абсолютно іншому образі. Знання таких елементів приходить тільки з досвідом. Потрібні фонові знання з історії підприємства, щоб зрозуміти професіоналізм повною мірою. Звичайно, така лексема перекладається, як правило, повним нейтральним еквівалентом [5: 9]. Перекладач повинен звертати пильну увагу на контекст вживання слів і виразів, щоб уникнути помилок і не потрапити в пастку «хибних друзів перекладача» [4: 33–36].

Під час перекладу професіоналізмів, найчастіше використовуються такі способи перекладу: словниковий відповідник, варіантний відповідник, транскодування, калькування, контекстуальна заміна, описовий переклад, перекладацький коментар.

З точки зору практики перекладу, всі елементи денотативної системи вихідної мови (лексичні та фразеологічні одиниці) поділяються на дві групи: ті, що вже мають відповідники («перекладацькі еквіваленти»), та ті, що не мають відповідників у мові перекладу [30: 720].

Словниковий відповідник – еквівалентні лексичні та фразеологічні одиниці не повністю представлені в перекладних словниках та в текстах, але існують в мові перекладу як перекладні еквіваленти. Неоднозначні слова мають кілька перекладацьких відповідників відповідно до кількісті їхніх значень (лексико-семантичних варіантів). Перші називаються одиницями, що мають перекладацькі відповідники у мові перекладу, а другі – безеквівалентними одиницями. Еквівалентні одиниці поділяються на одноеквівалентні (ті, що мають тільки один перекладацький відповідник) і багатоеквівалентні (ті, що мають два або більше перекладацьких відповідників). Слід мати на увазі, що йдеться про словникові відповідники, тоді як відповідник певного слова чи фрази оригіналу в тексті може бути тільки один з кількох. Еквівалентні лексичні одиниці не повністю представлені в двомовних словниках та в текстах, але існують в мові перекладу як перекладацькі еквіваленти [30: 720]. Наприклад: *network* ‘мережа’; *error* ‘помилка’; *lock* ‘блокування’.

Варіантний відповідник це відповідники неоднозначного слова називаються варіантними відповідниками. під варіантним відповідником розуміється один із можливих варіантів перекладу слова. Варіантний відповідник передає, як правило, якесь одне значення слова вихідної мови, тобто кожний варіантний відповідник є перекладним еквівалентом якогось одного лексико-семантичного варіанта багатозначного слова. Відповідно,

кожний з цих лексико-семантичних варіантів має свій перекладний еквівалент [63: 85]. Наприклад: *cloud storage* ‘хмарне сховище’*; device* ‘пристрій’.

Але перекладачі не завжди мають справу тільки із словниковими варіантними відповідниками – трапляється так, що словники не містять деяких відповідників багатозначного слова або ж словникові варіанти-відповідники певного слова взагалі не зафіксовані в словниках [30: 720].

Транскодування (передача «без перекладу») – звукова та/або графічна форма слова вихідної мови передається засобами абетки мови перекладу:

1) транскрипція – передбачає передачу професіоналізму у мову перекладача записом її фонетичного звучання, наприклад: *adapter* ‘адаптер’; *browser* ‘браузер’; *site* ‘сайт’;

2) транслітерація – спосіб перекладу професіоналізмів оригіналу шляхом відтворення її графічної форми мовою перекладу , наприклад: *buffer* ‘буфер’; *server* ‘сервер’; *router* ‘роутер’ [50: 47].

Калькування – цей метод полягає в буквальному перекладі елементів слова з мови-продуцента на мову-реципієнт. Як прийом перекладу частіше застосовується підчас перекладу складних слів [30: 722]. Калькування може застосовуватись також стосовно тільки одного з компонентів складного слова, тільки за умови збереження перекладним відповідником норми вживання і сполучуваності слів в мові перекладу. Наприклад: *semiconductor* ‘напівпровідник’; *data* ‘дані’ [20: 569].

Контекстуальна заміна – це така лексична перекладацька транс-формація, внаслідок якої перекладним відповідником стає слово або словосполучення, що не є словниковим відповідником, і що підібрано із врахуванням контекстуального значення слова, контексту вживання та мовленнєвих норм і традицій мови перекладу. Слід зазначити, що не існує точних правил створення контекстуальних замін, оскільки переклад слів у таких випадках залежить від контексту їхнього вживання [30: 722]. Видами

такої заміни є: конкретизація (слово ширшої семантики передається словом вужчої), генералізація (зворотній процес), антонімічний та описовий переклад, а також смисловий розвиток. Можливі також додавання слів, перестановка основ або навіть вилучення однієї з них. Наприклад: *constant* ‘постійна величина’ [64: 559].

Описовий переклад або дескриптивна перифраза – полягає в розкритті значення реалії мови-оригіналу за допомогою розгорнутих пояснень, що розкривають сутність позначуваного явища [50: 47].

До описового перекладу висуваються такі вимоги:

1) переклад повинен точно передавати основний зміст позначеного неологізмом поняття;

2) опис не повинен бути надто докладним;

3) синтаксична структура словосполучення не повинна бути складною [20: 570].

При застосуванні описового перекладу важливо слідкувати затим, щоб словосполучення в мові перекладу точно і повно передавало всі основні ознаки поняття, позначеного словом оригіналу. Порівняно з транскодуванням, завдяки описовому перекладу досягається більша прозорість змісту поняття, позначеного відповідником слова мови оригіналу [30: 723]. Наприклад, *Jimmy* ‘недосвідчений розробник програмного забезпечення, який думає, що все знає’; *duck* ‘позначка, зроблена з єдиною метою привернення уваги до себе зі сторони керівництва для подальшого її видалення без необхідності проведення жодних змін в програмі’; *bug* ‘вірус, який не можливо вивести, помічений тільки однією людиною’; *hot potato* ‘так жартома називають http:// та https://’; *hydra* ‘помилка, при видаленні якої з’являється дві інших; помилка, виправити яку неможливо’; *mad girlfriend bug* ‘коли ви бачите, що відбувається щось дивне, але програмне забезпечення каже, що все гаразд’; *ungoogleable* ‘людина, про яку немає інформації в інтернет-пошукачах, зокрема в Google’ [46: 274].

У тих випадках, коли важливим є не саме слово, а те значення, яке воно має в контексті оригіналу, перекладач застосовує **перекладацький коментар.** Цей прийом полягає в більш докладному, ніж опис, поясненні того, що означає вихідне слово в широкому контексті оригіналу. Таким чином, відтворюючи неперекладні елементи, перекладацькі коментарі виконують функцію інформування, ознайомлення читача з певними елементами, сприяють поглибленому та полегшеному їх розумінню [28: URL]. *There are two types of people: dummies and lamers. The former grow into gurus, while the latter remain lamers* ‘Є два типи людей: чайники* та ламери**. З перших виростають гуру, а другі так і залишаються ламерами’.

_____

*Новачок, недосвідчений користувач.

**Недосвідчена людина, яка не розуміється на певній сфері, але думає, що розуміється. Не плутати з чайником.

В. Н. Крупнов [38] наголошує, що застосування таких лексичних прийомів, як транслітерація, калькування, опис та пояснення належать до ефективних способів перекладу слів вузькоспеціалізованої лексики, зокрема, професіоналізмів. Найдоцільніше при перекладі професіоналізмів використовувати описовий метод оскільки професіоналізми не входять до складу загальновживаної лексики. За допомогою буквального перекладу, неможливо відобразити смислове навантаження та емоційне забарвлення таких слів [38: 137].

При перекладі професіоналізмів частіше використовуються варіанти розширеного перекладу – опису, перекладацький коментар, оскільки це дозволяє більш зрозуміло викласти смисловий зміст специфічного для певної сфери діяльності слова або вислові, не зрозумілих для широкої аудиторії. Точний переклад тут недоречний. Іноді для перекладу принципово нових професіоналізмів, що позначають технічне пристосування, або процес, можуть застосовуватися транскрипція і транслітерація, однак, при цьому вона може бути доповнена перекладацьким коментарем, або невеликим описом даного

об’єкта або процесу. У деяких випадках, загальновживане слово в певному контексті використовується для позначення предмета, який воно не означає в широкому сенсі, тоді таке слово також може бути віднесено до професіоналізму [4: 35].

Переклад професіоналізмів є складним, багатоступеневим процесом, що вимагає глибокого вивчення специфіки галузі використання професійних термінів, контексту та приватних значень слів та виразів. При перекладі професіоналізмів неприпустиме використання буквального перекладу, необхідно застосовувати різні перекладацькі прийоми, у тому числі транскрипцію та транслітерацію окремих слів, наприклад, технічних пристроїв, процесів, специфічних для даної професійної галузі. Переклад професіоналізмів не може бути здійснений коректно, без наявності у перекладача знань із конкретної сфери діяльності, що дозволяють йому орієнтуватися в термінах та реаліях, без наявності у нього відповідних словників та глосаріїв, що висвітлюють специфічну лексику, використовувану людьми однієї сфери діяльності. Саме необхідність орієнтуватися в суто професійній лексиці і робить переклад професіоналізмів настільки важким і водночас затребуваним завданням [43: 59].

Отже, головною метою будь-якого перекладу є досягнення адекватності. Адекватний переклад – це відтворення єдності змісту і форми оригіналу засобами іншої мови. Професіоналізми є елементами, що становлять додаткову складність при перекладі. Під час перекладу професіоналізмів найчастіше використовуються такі способи перекладу: словниковий відповідник, варіантний відповідник, транскодування, калькування, контекстуальна заміна. При перекладі професіоналізмів доречніше використовувати варіант розширеного перекладу – опису, оскільки це дозволяє більш зрозуміло викласти смисловий зміст специфічного для певної сфери діяльності слова або вислові, незрозумілих для широкої аудиторії.

1.3 Репрезентація діяльності з програмування у дискурсі інформаційних технологій

Професійне мовлення реалізується в користуванні мовою конкретної галузі в усній і писемній формах. Кінцевим результатом для фахівця стає не лише текст, а й дискурс, тобто сукупність мовленнєво-мисленнєвих дій комунікантів, пов’язаних із пізнанням, осмисленням та презентацією світу мовцем [27: 304].

Проблема визначення поняття «дискурс» є надзвичайно актуальною на сучасному етапі розвитку філологічної думки, ще досі вчені не вивели єдиного тлумачення цього складного міждисциплінарного феномена. Лінгвісти пропонують також різні типології дискурсу, але єдиної класифікації дискурсивних типів на даний момент не існує, адже не всі вони мають чіткі критерії та достатню кількість емпіричного матеріалу [41: 147].

Детальне вивчення явища «дискурсу» вказує на різні тлумачення різними авторами. Зокрема, знаходимо наступне розуміння дискурсу: послідовність мовленнєвих актів, тобто текст, включений у комунікативно-прагматичний контекст, на відміну від тексту як послідовності речень, відстороненої від комунікативно-прагматичного контексту [15: URL].

Ще одним різновидом інтерпретації терміну «дискурс» є такий, що обумовлений прагматичною диференціацією всього дискурсивного масиву мови і тому часто замінюється на термін «дискурсивні практики». За таким тлумаченням, дискурс є комунікативно-прагматичним зразком мовленнєвої поведінки, котра протікає в певній соціальній сфері і має певний набір перемінних, якими слугують сталі соціальні норми, соціальні відношення між комунікантами, ролі, які ці комуніканти виконують, чинні конвенції, показники інтерактивності комунікантів тощо [16: 188]

На думку В. Є. Чернявської [65], дискурс означає:

1) конкретну комунікативну подію, фіксовану в письмових текстах і усному мовленні, здійснювану в певному, когнітивно і типологічно зумовленому комунікативному просторі;

2) сукупність тематично співвідносних текстів: тексти, що об'єднані в дискурс, звернені, так чи інакше, до однієї загальної теми [65: 14–16].

За дефініцією Ю. С. Степанова [59], дискурс – це можливий альтернативний світ у світі мови, що має «особливу граматику, особливий лексикон, особливу семантику, особливі правила слововживання» [59: 42]. Тобто, спостерігається прагнення до розмежування індивідуального мовлення як конкретного індивідуального акту висловлювання думки певним мовцем із характерним саме для цього мовця набором лексичних, граматичних, стильових та інших особливостей оформлення власної думки, і узагальненого, типологічного зразка мовленнєвої поведінки, притаманної певній конкретній комунікативній ситуації [16: 188].

Широке тлумачення дискурсу подає О. О. Селіванова [56]: **дискурс** – це передусім зв’язний текст у контексті різних факторів – соціокультурних, психологічних та ін.; це текст, занурений у життя. В іншому аспекті дискурс виявляє себе як цілісність комунікативної ситуації (подія), у яку занурений текст, і комуніканти, які формуються на основі різноманітних факторів (соціальних, культурних, етнічних та ін.), опосередковуючи спілкування і розуміння. Дискурсом дослідниця називає і мову, переважно усну, стиль, підмову спілкування, а також зразок мовної поведінки, яка характерна для певної соціальної сфери і має свій набір змінних [56: 298].

Кожна сфера людської діяльності породжує свій тип дискурсу, з чого можна прийти до висновку, що дискурс є тим феноменом, який дуже складно простежити і проаналізувати, адже він має багато вимірів і є утворенням лабільним [10: 61]. У будь-якій сфері діяльності людини створення і широке використання технологій веде до посилення можливостей та підвищення

потенціалу цієї сфери, а також до опосередкованого розвитку інших суміжних сфер людської діяльності. Таким чином, технологія являє собою процес послідовного, ступеневого здійснення, розробленого на науковій основі вирішення певної виробничої або соціальної проблеми [2: 43].

Інформатизація суспільства, стрімкий технологічний прогрес призвели до появи та розвитку нових способів спілкування та комунікації. Одним із виразних атрибутів сучасності є динамічний розвиток засобів спілкування, що демонструє народження нових форм інформаційного обміну в віртуальному середовищі – просторі, який сконструйований на основі цифрових технологій. В віртуальному середовищі співрозмовці продукують деякі зразки, норми та правила мовленнєвої взаємодії, модифікуючи при цьому процес комунікації в його класичному, традиційному враженні [15: URL].

Сучасність характеризується інтенсивним розвитком інформаційних технологій та формуванням інформаційного суспільства. Кожна особа, як частина інформаційного суспільства, піддається впливу прогресу інформаційних технологій, який в умовах глобалізації світової спільноти, безсумнівно, призводить до породження і розвитку зовсім нових способів спілкування. Нові способи спілкування, у свою чергу, набувають нових форм реалізації – віртуальних або електронних/цифрових в сучасному інформаційному суспільстві. Інформаційні технології є факторами впливу як на чат-комунікації, так і на живі комунікації, однак мова і мовлення в Інтернеті підлягає подібному впливу, так само як в реальному середовищі [15: URL].

Інформація (від лат. *informatio* ‘роз’яснення, виклад’) – відомості, що передаються людьми усним, письмовим або іншим способом (за допомогою умовних сигналів, технічних засобів тощо); одне з основних понять кібернетики, це будь-який вид знань, якими можуть обмінюватися користувачі, про речі, факти, поняття і так далі, у всесвіті дискурсу, вона повинна бути виражена в певній формі подання, для того щоб вона була зрозумілою для певного кругу осіб [14: 11].

Технологія, в загальному розумінні, це сукупність методів, засобів і реалізації людьми конкретного складного процесу шляхом поділу його на систему послідовних взаємопов’язаних процедур і операцій, які використовуються більш або менш однозначно і мають на меті досягнення високої ефективності певного виду діяльності [8: 18].

Важливим для цього дослідження є також поняття «інформаційні технології». Зокрема, О. П. Буйницька [8] дає таке трактування: інформаційні технології – сукупності методів, засобів і прийомів, що використовуються для забезпечення ефективної діяльності людей в різноманітних виробничих і невиробничих сферах [8: 19].

За Т. В. Янчуком [69] інформаційні технології – це комплекс взаємозалежних, наукових, технологічних, інженерних дисциплін, що вивчають методи ефективної організації праці людей, зайнятих обробкою і зберіганням інформації; обчислювальну техніку та методи організації і взаємодії з людьми та виробничим устаткуванням, практичні додатки, а також пов'язані з усім цим соціальні, економічні і культурні проблеми. Самі інформаційні технології вимагають складної підготовки, великих початкових витрат і наукомісткої техніки [69: 270], маємо визначити, це найбільш вдала дефініція.

О. В. Андрощук [2] дає таку влучну дефініцію: **інформаційна технологія** – це процес, що використовує сукупність засобів і методів збору, накопичення, обробки і передачі даних(первинної інформації) для отримання оновлених даних про стан об'єкта, процесу або явища (інформаційного продукту). Цей процес складається з чітко регламентованої послідовності виконання операцій, дій, етапів різного ступеня складності з даними, які зберігаються на комп'ютерах. Основною метою інформаційної технології є обробка первинної інформації з допомогою цілеспрямованих дій та отримання необхідної користувацької інформації, а також отримання та обробка даних для їх подальшого аналізу і прийняття рішень про виконання певної дії [2: 43].

Тож інформатизація суспільства ґрунтується на інформаційних технологіях, впровадження яких потребує відповідного інтелектуального забезпечення – інформаційні технології стають стратегічним чинником прогресу. Виокремлюється формування нових форматів мовленнєвої діяльності через створення нових засобів інформаційних технологій, адже в постійно поновлюваному суспільстві зростає вага інформації як стратегічного засобу передових громадських трансформацій, а інформація стає потужним всевладним ресурсом [15: URL]. Вплив сучасних інформаційних технологій породжує проблему віртуалізації ціннісних орієнтацій сучасної людини, або віртуалізації реальності, а також формування власне віртуальної реальності. Дана проблема набула значної актуальності, а дискурс її обговорення поширений на різноманітні наукові дисципліни: від кібернетики до філософії [18: 63].

Діяльність з інформатизації охоплює:

1) комп’ютерне програмування, консультування із питань інформатизації, діяльність із керування комп’ютерним устаткуванням;

2) видання: програмного забезпечення та комп’ютерних ігор;

3) надання програмних продуктів у режимі «онлайн»;

4) навчання (комп’ютерній грамотності, розробленню, модифікації, тестуванню та технічній підтримці програмного забезпечення, бізнес-аналізу), організацію графічних інтерфейсів, процесів із контролю якості, системного адміністрування, проєктного менеджменту, розроблення документацій;

5) оброблення даних, розміщення інформації на веб-вузлах і пов’язану з ними діяльність, веб-портали;

6) дослідження та експериментальні розробки у сфері інформаційних та інформаційно-комунікаційних технологій [22: 43].

У зв’язку з масовою комп’ютеризацією всіх видів діяльності людини увагу мовознавців привертає комунікація у сфері інформаційних технологій. Інформаційні технології застосовуються в політичних, наукових, економічних

та культурних галузях і мають відповідні дискурси зі своїми особливостями. Ці дискурси здебільшого належать до інституційного типу, хоча необхідно пам’ятати, що інституційний дискурс як і будь-які інші типи дискурсу не ставить строгі вимоги щодо особистісної орієнтованості; в різних дискурсах вона різна. Всі інституційні дискурси є професійними, але не всі професійні дискурси є інституційними [40: 41–42].

Формально англомовний **дискурс програмування** у його письмовій формі – це монолог, актуалізація на письмі мови автора, з погляду прагматики – діалог, проте, не автора із самим собою, а автора з потенційним читачем, точніше уявним співрозмовником, як типовим представником тієї соціально-культурної спільноти, якій автор адресує своє повідомлення [11: 85].

Професійний дискурс програмістів має такі особливості:

1) приналежність до діалогічного дискурсу;

2) віднесеність дискурсу, як правило, до інструментального типу, тобто, коли відбувається обмін думками з того чи іншого приводу;

3) особлива лексика, що складається з таких лексичних пластів, як, літературні слова, терміни, сленг, розмовні та просторічні фрази та висловлювання, основу яких становлять, в більшості випадків, англо-американські запозичення [34: 596–597].

Таким чином, професійний дискурс програмістів представляє досить цікаве явище в лінгвістиці та викликає великий інтерес у зв'язку з глобальної комп’ютеризацією нашого суспільства та величезної популярності комп’ютерної лексики [34: 597].

Автор текстів англомовного дискурсу програмування – не лише суб’єкт пізнання та перетворення дійсності, але й мовна особистість, що володіє сукупністю вербальних характеристик, які обумовлюють її здатність створювати та сприймати тексти, що відзначаються різним ступенем структурно-мовної складності, глибини й точності відображення дійсності.

Основними мовними засобами вираження експліцитної присутності автора у тексті англомовного дискурсу програмування є ті конструкції, у яких на першому місці стоїть вказівка на автора [11: 86]. Наприклад, автор висловлюється від власного імені, використовуючи авторське *I* ‘я’ (або *we* ‘ми’, якщо книгу написано у співавторстві*): I tend to add a meaningless primary key to migrate to other entities to help me see when there is any ownership* [73: 40]. *In this chapter, we will quickly introduce “hello world” pieces of code to give you hints on what you will discover in the rest of the book* [76: 33].

Так зване інклюзивне *we (our, us)* у цьому прикладі об’єднує у своїй семантиці як адресанта, так і адресата повідомлення. Автор тим самим піднімає читача до рівня свого союзника та рівноправного учасника процесу міркувань на теми, пов’язані з проектуванням та розробленням програмного забезпечення. В. Є. Чернявська [66] називає цей прийом створенням семантичного поля «свої» [66: 48]. Особовий займенник «ми» і відповідний йому присвійний займенник «наше» стають розширеним позначенням відправника повідомлення, за допомогою якого він об’єднує себе в адресатній функції з іншими особами [11: 86]. Як наслідок, створюється ефект колективного автора повідомлення, яке ідентифікує єдину позицію обох учасників комунікативного процесу, зближує плани автора та читача [66: 48]. Автор висловлюється від власного імені та від імені узагальненого адресата, використовуючи займенник *we* ‘ми’, наприклад: *Dijkstra pointed out that no one’s skull is really big enough to contain a modern computer program (Dijkstra 1972), which means that we as software developers shouldn’t try to cram whole programs into our skulls at once; we should try to organize our programs in such a way that we can safely focus on one part of it at a time* [77: 117].

На думку О. Ю. Винник та Н. В. Рубель [11], основним персуазивним засобом впливу на читача в англомовному дискурсі програмування є спонукальні речення, оскільки їх роль полягає у спонуканні адресата до практичного застосування інформації й активної співпраці з автором, спрямованої на

досягнення комунікативних цілей читача – набуття чи удосконалення знань і навичок програмування [11: 87]: *Now put this into a test harness, TestHookTemplate.java, and call thecookieHookRobot.gomethod* [73: 160].

Професійний дискурс інженера-програміста представляє маловивчене унікальне явище в сучасному мовознавстві та викликає чималий інтерес у зв'язку з глобальною інформатизацією суспільства та великою популярністю інформаційних технологій. Найбільшу особливість професійного дискурсу інженера-програміста становить різноманітна в семантичному аспекті спеціальна лексика, що включає слова професіоналізми [49: 163].

При цьому важливо розуміти, що однією з важливих функцій професійного дискурсу програміста є передача емоцій, і, насамперед, це відноситься до функціонування професіоналізмів. Наприклад, іронічний характер відчувається у таких словах, як: *handshake* ‘метод керування синхронною передачею даних до повільного пристрою, такого як принтер, коли кожна операція передачі вимагає сигналу підтвердження’; *candidate* ‘перспектива’, *beauty* ‘перевага’. Такі слова допомагають пожвавити професійний дискурс, відвернути програмістів від складної та монотонної роботи, тому вони вводять у свій індивідуальний лексикон – такі одиниці [49: 163].

Стратегія залучення читача до співпраці з автором у сучасному англомовному дискурсі програмування реалізується за допомогою свідомої імітації автором природного діалогу у парі «автор – читач». Різноманітні структури діалогізації та риторичні прийоми, серед яких прямі звертання до читача, умовні, спонукальні, питальні речення, надають текстам англомовного дискурсу програмування характеру дружньої бесіди автора з читачем, сприяють інтимізації викладу, створенню атмосфери довіри між комунікантами [11: 87].

Отже, професійне мовлення реалізується в користуванні мовою конкретної галузі в усній і писемній формах. Дискурс є тим феноменом, який

дуже складно простежити і проаналізувати, адже він має багато вимірів і є утворенням лабільним. Професійний дискурс програмістів представляє досить цікаве явище в лінгвістиці та викликає великий інтерес у зв'язку з глобальної комп'ютеризацією нашого суспільства та величезної популярності комп'ютерної лексики. Інформаційна технологія – це процес, що використовує сукупність засобів і методів збору, накопичення, обробки і передачі даних для отримання оновлених даних про стан об'єкта, процесу або явища. Інформатизація суспільства ґрунтується на інформаційних технологіях, впровадження яких потребує відповідного інтелектуального забезпечення – інформаційні технології стають стратегічним чинником прогресу.

Висновки до розділу 1

1. Мовою для спеціальних цілей називають сукупність усіх мовних засобів, які застосовуються в певній обмежуваній фахом галузі комунікації для забезпечення взаєморозуміння між зайнятими в цій галузі фахівцями. Спеціальна лексика – це слова чи словосполучення, що позначають поняття спеціальної галузі знання або діяльності, в якій у процесі спілкування використовуються професіоналізми (напівофіційні слова, сферою уживання яких є, як правило, усне побутово-професійне мовлення). Вони виникають у двох випадках: коли певна галузь не має розвиненої термінології, або як неофіційні замінники термінів, які вживають у розмовній мові.

2. Професіоналізми вважаються складною категорією з точки зору перекладу, оскільки сфера їх вживання досить обмежена і вони не зафіксовані в словниках. Застосування таких лексичних прийомів, як транслітерація, калькування, семантична модифікація, опис та пояснення належать до ефективних способів перекладу слів вузькоспеціалізованої лексики, зокрема, професіоналізмів. Переклад професіоналізмів є складним, багатоступеневим процесом, що вимагає глибокого вивчення специфіки галузі використання

професійних термінів, контексту та приватних значень слів та виразів. Найдоцільніше при перекладі професіоналізмів використовувати описовий метод або метод коментування, оскільки професіоналізми не входять до складу загальновживаної лексики.

3. Нові способи спілкування, у свою чергу, набувають нових форм реалізації – віртуальних або електронних / цифрових в сучасному інформаційному суспільстві. У зв’язку з масовою комп’ютеризацією всіх видів діяльності людини увагу мовознавців привертає комунікація у сфері інформаційних технологій, впровадження яких потребує відповідного інтелектуального забезпечення – інформаційні технології стають стратегічним чинником прогресу. Інформатизація суспільства ґрунтується на інформаційних технологіях, їх вплив породжує проблему віртуалізації ціннісних орієнтацій сучасної людини, або віртуалізації реальності, а також формування власне віртуальної реальності. Професійний дискурс програмістів є досить цікавив явищем в лінгвістиці та викликає великий інтерес у зв'язку з глобальною комп'ютеризацією суспільства та величезної популярності комп’ютерної лексики. Найбільшу особливість професійного дискурсу інженера-програміста становить різноманітна в семантичному аспекті спеціальна лексика, що включає слова професіоналізми.

РОЗДІЛ 2
ЛІНГВІСТИЧНІ ПАРАМЕТРИ ПРОФЕСІОНАЛІЗМІВ У КОНТЕКСТІ ПРОГРАМУВАННЯ ІТ ДИСКУРСУ: НА ПРИКЛАДІ ПРОГРАМУВАННЯ МОВОЮ JAVA

2.1 Семантичні параметри професіоналізмів у контексті програмування ІТ дискурсу

У ході дослідження професіоналізми ІТ дискурсу, виявлені у текстах, присвячених мові програмування Java, розподілено на тематичні групи і підгрупи. Зокрема, виділено такі основні тематичні групи, як «людина», «процес програмування», «програмне забезпечення», «апаратне забезпечення» та «суміжні науки». Розглянемо більш докладно наповнення наведених тематичних груп.

У тематичній групі «**людина**» виділено дві підгрупи:

1) «**учасники процесу програмування**»: (Додаток А, приклад 8, далі зазначається номер прикладу) *seasoned developer* «людина, яка має великий досвід у розробці програмного забезпечення» (ТР: URL);

2) «**характеристики учасників процесу програмування**»: (84) *bad programming style* «коли програміст створює програму, щоб швидше виконувати завдання, не замислюючись про майбутні зміни та ігноруючи можливість інших розробників торкатися коду» (ТР: URL).

Тематична група «**процес програмування**» налічує п’ять основних підгруп:

1) «**мова програмування**», зокрема:

а) «**мови програмування взагалі**»: (59) *object-oriented programming language* «мова, що використовується в моделі комп’ютерного програмування, яка організовує дизайн програмного забезпечення навколо даних або об’єктів, а не функцій і логіки» (ТР: URL); (61) *object-oriented programming* «модель

комп’ютерного програмування, яка організовує дизайн програмного забезпечення навколо даних або об’єктів, а не функцій і логіки» (TP: URL); (69) *general-purpose programming* «мова програмування, призначена для створення програмного забезпечення в широкому спектрі прикладних областей, у безлічі апаратних конфігурацій і операційних систем» (TP: URL); (81) *statically-typed* «характеристика мови програмування, в якій типи змінних оголошуються явно і, таким чином, визначаються під час компіляції» (TP: URL);

б) «**складові мов програмування**»:

– «**класи**»: (6) *class* «шаблон, який використовується для створення об’єктів і визначення типів і методів даних об’єктів» (TP: URL); (65) *superclass* «клас, з якого можна створити багато підкласів, при чому підкласи успадковують характеристики суперкласу» (TP: URL);

– «**дані**»: (38) *ID (identifier)* «лексичний токен, що називає сутність» (CD: URL); (55) *namespace* «декларативна область, яка надає область ідентифікаторів (назви типів, функцій, змінних тощо) всередині неї; використовується для організації коду в логічні групи та для запобігання конфліктам імен, які можуть виникнути, особливо якщо ваша база коду містить кілька бібліотек» (TP: URL); (87) *primitive types* «стандартні типи даних, вбудовані в мову програмування (типи даних, які не є примітивними, називаються похідними або складеними); примітивні типи майже завжди є типами значень, але складені типи також можуть бути типами значень» (TP: URL); (88) *literal* «значення, які записуються в умовній формі, але які є очевидними» (TP: URL);

– «**бібліотеки**»: (9) *library* «збірка класів, які вже були написані кимось іншим» (TP: URL); (68) *class library* «колекція попередньо написаних класів або закодованих шаблонів, будь-який з яких може бути визначений і використаний програмістом під час розробки прикладної програми» (TP: URL);

– «**змінні**»: (13) *environment variable* «динамічне значення, яке може впливати на поведінку запущених на комп’ютері процесів» (TP: URL); (39) *scratch variable* «змінна, яку можна змінити або отримати доступ з будь-якого спрайту в проєкті або етапі, незалежно від того, на якому спрайті її було створено» (CD: URL); (74) *non-static field* «будь-які оголошені змінні класу, які не є статичними» (TP: URL); (75) *local variable* «змінна, якій надається локальна область» (TP: URL);

– «**вирази**»: (7) *declaration* «визначення імені та типу даних змінної або іншого елемента» (TP: URL); (22, 56) *classpath* «параметр у віртуальній машині Java або компіляторі Java, який визначає розташування визначених користувачем класів і пакетів» (TP: URL); (29) *import statement* «вираз у мові програмування, що може використовуватися для імпорту цілого пакета або іноді для імпорту певних класів та інтерфейсів усередині пакета» (TP: URL); (66) *class declaration* «компонент оголошення у програмному коді, який оголошує ім'я класу разом з іншими атрибутами, такими як суперклас класу, і те, чи є клас публічним, остаточним або абстрактним» (TP: URL); (80) *reserved word* «слово, яке не можна використовувати як ідентифікатор, наприклад ім’я змінної, функції чи мітки – воно “зарезервовано для використання”» (TP: URL); (92) *precedence* «правило, яке визначає порядок, у якому певні операції повинні виконуватися у виразі» (TP: URL);

– «**команди**»: (36) *subcommand* «команда, яка є частиною більшої команди» (CD: URL); (91) *operator* «символ, який зазвичай представляє дію або процес» (TP: URL); (93) *assignment operator* «правило, яке визначає порядок, у якому певні операції повинні виконуватися у виразі» (TP: URL); (94) *object references* «адреса, яка вказує, де зберігаються змінні та методи об’єкта» (TP: URL); (95) *unary operator* «оператор, який приймає один операнд у виразі або операторі» (TP: URL); (96) *ternary operator* «оператор, який існує в деяких мовах програмування, який приймає три операнди, а не типові один або два, які використовують більшість операторів» (TP: URL);

2) «**програмний код та його складові**»:

а) «**програмний код**»: (3) *source code* «набір комп'ютерних інструкцій, які були написані для створення програми або частини програмного забезпечення» (CD: URL); (10) *build* «версія частини програмного забезпечення, комп'ютерної гри, веб-сайту тощо» (CD: URL); (76) *exception handler* «код, який визначає, що програма робитиме, коли аномальна подія порушує нормальний потік інструкцій цієї програми» (TP: URL); (17) *executable code* «будь-який об'єктний, машинний або інший код, який зчитується комп'ютером при завантаженні в його пам'ять і безпосередньо використовується таким комп'ютером для виконання інструкцій» (TP: URL);

б) «**складові програмного коду**»: (30) *snippet* «невелика область багаторазового вихідного коду, машинного коду або тексту» (TP: URL); (37) *method call* «метод, який містить фактичний виклик» (CD: URL);

в) «**функції**»: (43) *nonfinal feature* «функція, що використовується для виклику функцій конкретних класів в абстрактному класі» (TP: URL); (44) *switch expression* «тип механізму керування вибором, який використовується, щоб дозволити значенню змінної або виразу змінити потік керування виконанням програми за допомогою пошуку та відображення» (TP: URL); (45) *Z garbage collector* «масштабований збирач програмного сміття з низькою затримкою, який виконує всю роботу одночасно, не зупиняючи виконання потоків додатків більше ніж на 10 мс, що робить його придатним для додатків, які потребують низької затримки та/або використовують дуже велику купу (кілька терабайтів)» (TP: URL); (46) *incubating feature* «експериментальні API, що розповсюджуються у вигляді окремих модулів з іменами з префіксом "jdk. incubator"» (TP: URL);

3) «**стадії програмування**»: (21) *encapsulation* «об'єднання даних з методами, які працюють з цими даними, або обмеження прямого доступу до деяких компонентів об'єкта з метою приховати значення або стан об'єкта структурованих даних у класі, запобігаючи прямому доступу до них клієнтами

таким чином, щоб відкрити приховані деталі реалізації або порушити інваріантність стану, що підтримується методами» (TP: URL); (42) *incubating* «фаза програмування, яка полягає у створенні повністю функціонуючого проекту з відкритим кодом» (TP: URL); (49) *iteration* «процес, у якому набір інструкцій або структур повторюється в послідовності задану кількість разів або до виконання умови» (TP: URL); (63) *to debug* «багатоетапний процес, який включає виявлення проблеми, ізоляцію джерела проблеми, а потім або виправлення проблеми, або визначення способу її вирішення» (TP: URL);

4) «**складові процесу програмування**»:

а) «**програмне забезпечення для програмування**»: (2) *compiler* «комп'ютерна програма, яка передає інструкції машинною мовою» (CD: URL); (18) *Integrated Development Environment* «програмний додаток, який надає програмістам комплексні можливості для розробки програмного забезпечення; зазвичай складається принаймні з редактора вихідного коду, інструментів автоматизації збірки та налагоджувача» (TP: URL); (89) *constructor* «спеціальний метод класу або структури в об'єктно-орієнтованому програмуванні, який ініціалізує новостворений об'єкт такого типу; щоразу, коли створюється об'єкт, конструктор викликається автоматично» (TP: URL); (90) *initializer* «компонент програми, що запускає процес пошуку та використання визначених значень для змінних даних, які використовуються комп'ютерною програмою» (TP: URL); (47) *core platform* «центр системи промислової автоматизації Asset to Enterprise, яка збирає та перерозподіляє дані до відповідних інженерних, експлуатаційних і технічних програм» (TP: URL);

б) «**сценарії виконання програм**»: (16) *error message* «повідомлення, яке відображається на екрані монітора або роздруківці, яке вказує на те, що було надано неправильну інструкцію або що сталася помилка через несправне програмне чи апаратне забезпечення» (TP: URL); (25) *startup script* «файл, який містить команди, що виконуються під час завантаження екземпляра

віртуальної машини (VM)» (TP: URL); (26) *script* «програма або послідовність інструкцій, які інтерпретуються або виконуються іншою програмою, а не процесором комп'ютера (як скомпільована програма)» (CD: URL);

в) «**характеристики програмованого продукту**»: (40) *critical workload* «спосіб думати про обсяг обробки, який сервер повинен буде виконати за фіксований період часу» (CD: URL); (51) *backward-compatibility rule* «властивість апаратної або програмної системи, яка може успішно використовувати інтерфейси та дані з попередніх версій системи або з іншими системами» (TP: URL); (78) *auto-generated* «такий, що не був написаний особою, а був згенерований автоматизованим процесом» (TP: URL); (82) *immutable* «характеристика об'єкту, стан якого не можна змінити після його створення» (TP: URL); (97) *bitwise* «рівень роботи, який передбачає роботу з окремими бітами, які є найменшими одиницями даних в обчислювальній системі» (TP: URL);

5) «**дії в процесі програмування**»:

а) «**дії**»: (1) *to run* «виконати програму» (CD: URL); (19) *to nail down bug* «виокремити помилки в програмному коді з метою їх виправлення» (TP: URL); (34) *to override* «функція об'єктно-орієнтованого програмування, яка дозволяє дочірньому класу забезпечувати іншу реалізацію для методу, який уже визначено та / або реалізовано в його батьківському класі або одному з його батьківських класів» (TP: URL); (35) *to set to the default* «навмисно чи випадково дозволити програмі використовувати попередньо встановлене значення або налаштування для елемента» (TP: URL); (50) *test-bed mechanism* «метод тестування окремого модуля (функції, класу або бібліотеки), що можна використовувати як доказ концепції або коли новий модуль тестується окремо від програми / системи, і його буде додано пізніше» (TP: URL); (57) *to revamp* «надати нову та вдосконалену форму, структуру або зовнішній вигляд» (TP: URL); (60) *data encapsulation* «є однією з основ об'єктно-орієнтованого програмування, стосується об'єднання даних із методами, які працюють із

цими даними і використовується для приховування значень або стану об’єкта структурованих даних у класі, запобігаючи прямому доступу до них неавторизованих сторін» (TP: URL); (62) *code re-use* «використання існуючого програмного забезпечення або знань про програмне забезпечення для створення нового програмного забезпечення» (TP: URL); (73, 85) *to initialize* «присвоєння початкового значення для об’єкта даних або змінної» (TP: URL); (52) *by default* «заздалегідь розроблене значення або параметр, який використовується комп’ютерною програмою, коли значення або параметр не вказано користувачем програми» (TP: URL); (83) *reasonable default* «ситуація, коли поля, які оголошені, але не ініціалізовані, будуть встановлені компілятором за умовчанням» (TP: URL); (98) *method invocation* «метод, коли клієнтський об’єкт запитує послугу (за описом), але не знає конкретного ідентифікатора об’єкта або класу об’єкта для задоволення запиту» (TP: URL);

б) «**оцінка ефективності дій**»: (53, 86) *compile time* «період, коли код програмування (наприклад, C#, Java, C, Python) перетворюється на машинний код (тобто двійковий код)» (CD: URL); (67) *build time* «час, витрачений на те, щоб зібрати щось, наприклад нову модель, або створити нову версію комп’ютерної програми» (CD: URL).

До складу тематичної групи «**програмне забезпечення**» належать такі підгрупи:

1) «**робота програмного забезпечення**»: (12) *to unzip* «розпакувати стиснутий файл» (CD: URL); (32) *to preload* «безкоштовна програма для Linux, яка працює як демон (програма, яка безперервно працює як фоновий процес і прокидається для обробки періодичних запитів на обслуговування, які часто надходять від віддалених процесів) для запису статистики про використання файлів програмами, які найчастіше використовуються» (TP: URL); (33) *to overwrite* «переписування або заміна файлів та інших даних у комп’ютерній системі чи базі даних новими даними» (CD: URL); (79) *self-documenting* «веб-сайт, який розкриває весь процес його створення через публічну

документацію, і чия публічна документація є частиною процесу розробки» (TP: URL);

2) «**взаємодія програми з користувачем**»:

а) «**засоби взаємодії**»: (14) *DOS prompt* «критична частина інтерфейсу командного рядка Microsoft Disk Operating System (MS-DOS)» (TP: URL); (54) *interface* «структура / синтаксис програмування, що дозволяє комп'ютеру застосовувати певні властивості до об'єкта (класу)» (CD: URL); (71) *graphical user interface* «комп'ютерна програма, яка дозволяє людині спілкуватися з комп'ютером за допомогою символів, візуальних метафор і вказівних пристроїв» (TP: URL); (72) *bookmark* «функція веб-браузера, яка використовується для збереження URL-адреси для використання в майбутньому» (TP: URL);

б) «**рівні взаємодії**»: (24) *language shell* «рівень програмування, що розуміє та виконує команди, які вводить користувач» (TP: URL);

в) «**способи взаємодії**»: (27) *I/O (input / output)* «будь-яка операція, програма або пристрій, який передає дані на або з комп'ютера» (TP: URL); (31) *verbose mode* «опція, доступна в багатьох комп'ютерних операційних системах і мовах програмування, яка надає додаткові відомості про те, що робить комп'ютер і які драйвери та програмне забезпечення він завантажує під час запуску або під час програмування, вона створить детальний вихід для діагностичних цілей, таким чином полегшує налагодження програми» (TP: URL); (41) *preliminary access* «модель фінансування в індустрії програмного забезпечення, за якою споживачі можуть придбати програму та використовувати її в різних циклах розробки перед випуском, наприклад пре-альфа, альфа та/або бета, тоді як розробник може використовувати ці кошти для продовження подальшого розвитку програми» (CD: URL); (58) *early access* «модель фінансування в індустрії програмного забезпечення, за якою споживачі можуть придбати програму та використовувати її в різних циклах розробки перед випуском, наприклад пре-альфа, альфа та/або бета, тоді як

розробник може використовувати ці кошти для продовження подальшого розвитку програми» (CD: URL); (64) *pluggability* «властивість програмного забезпечення, яка означає, що можна видалити модуль, можливо, замінивши на свій» (CD: URL); (77) *case-sensitive* «здатний розрізняти великі та малі версії літери в наборі символів мови» (TP: URL);

3) «**файлова система та типи файлів**»: (5) *plain text file* «тип цифрового файлу, який не містить комп'ютерних тегів, спеціального форматування та коду» (CD: URL); (11) *ZIP file* «стиснуті файли, які займають менше місця в пам'яті та можуть бути перенесені на інші комп'ютери швидше, ніж файли без стиснення» (TP: URL); (15) *distribution file* «файл, що використовується для переміщення продукту від джерела виробництва до споживачів» (TP: URL); (23) *shebang file* «файл, що містить послідовність символів, яка складається зі знаку числа та знаку оклику (#!) на початку сценарію» (TP: URL); (48) *source code repository* «архів із кодом, а також хостинг для цих архівів програмного забезпечення, де також можна тримати технічну документацію проекту, веб-сторінки, фрагменти, виправлення тощо, до яких можна отримати публічний (з відкритим кодом) або приватний доступ» (TP: URL).

Окремою тематичною групою постає група «**апаратне забезпечення**», що містить такі підгрупи:

а) «**обчислювальне обладнання**»: (20) *in-memory* «внутрішня пам'ять електронно-обчислювальної машини» (CD: URL); (100) *unit of execution* «частина центрального процесора (CPU), яка виконує операції та обчислення згідно з інструкціями комп'ютерної програми» (TP: URL);

б) «**мережеве обладнання**»: (28) *pipe chain* «труба, через яку кабельний ланцюг проходить від брашпиля до ланцюгової шафки» (TP: URL); (70) *network socket* «структура програмного забезпечення в мережевому вузлі комп'ютерної мережі, яка служить кінцевою точкою для надсилання та отримання даних через мережу» (TP: URL).

Група професіоналізмів «**суміжні науки**» містить такі професіоналізми: (99) *floating point arithmetic* «арифметика з використанням формульного представлення дійсних чисел як наближення для підтримки компромісу між діапазоном і точністю» (TP: URL); (4) *tutorial* «документ або веб-сайт, який показує, як використовувати продукт у серії простих етапів» (CD: URL).

Перелік тематичних груп і підгруп професіоналізмів IT дискурсу сфери програмування представлено у *Таблиці 2.1*.

Таблиця 2.1

Тематичні групи професіоналізмів у контексті програмування IT дискурсу

Тематичні групи / підгрупи	**Кількість**	**Частка**
1. Людина	**2**	**2%**
учасники процесу програмування	1	1%
характеристики учасників процесу програмування	1	1%
2. Процес програмування	**72**	**72%**
мова програмування	29	29%
програмний код та його складові	10	10%
стадії програмування	4	4%
складові процесу програмування	13	13%
дії в процесі програмування	16	16%
3. Програмне забезпечення	**20**	**20%**
робота програмного забезпечення	4	4%
взаємодія програми з користувачем	11	11%

Закінчення Таблиці 2.1

Тематичні групи / підгрупи	**Кількість**	**Частка**
файлова система та типи файлів	5	5%

4. Апаратне забезпечення	**4**	**4%**
обчислювальне обладнання	2	2%
мережеве обладнання	2	2%
5. Суміжні науки	**2**	**2%**
Загалом	**100**	**100%**

Графічна репрезентація семантичної структури професіоналізмів у контексті програмування ІТ дискурсу наведена в *Додатку Б* (найбільш чисельні групи і підгрупи помічено зеленим кольором, найменш чисельні – червоним; всередині підгрупи частотність професіоналізмів тієї чи іншої складової відзначено інтенсивністю кольору заливки).

Як видно з *Таблиці 2.1* та *Додатку Б*, професіоналізми у контексті програмування ІТ дискурсу найчастіше стосуються процесу програмування (72%), зокрема, дій у процесі програмування (16%) та складових процесу програмування (13%).

2.2 Особливості творення професіоналізмів у контексті програмування ІТ дискурсу

Професіоналізми в контексті програмування ІТ дискурсу творяться за різними моделями, як результат, вони представлені словами та словосполученнями. Розглянемо способи творення таких мовних одиниць у дискурсі інформаційних технологій, де розкриваються аспекти програмування.

1. Професіоналізми у формі **слів** утворюються за такими словотвірними моделями:

1) **афіксація**:

а) **префіксальний спосіб**:

– *un-* + *zip* «‘блискавка’, назва формату запакованих в архів файлів» → *unzip* (значення професіоналізмів див. у *п. 2.1*): (12) *You can unzip the content of this file anywhere on your computer* (LJ: URL);

– *sub-* + *command* «команда» → *subcommand*: (36) *When entering snippets, commands, subcommands, command arguments, or command options, use the Tab key to automatically complete the item* (LJ: URL);

– *inter-* + *face* «зовнішній бік» → *interface*: (54) *You need a first mandatory Square class, which you need to store in the same package or module as the Shape interface* (LJ: URL);

– *re-* + *vamp* «змити, очистити» → *revamp*: (57) *The goal of those projects is to conduct fundamental investigations in particular areas to drastically improve (or completely revamp) certain aspects of the Java platform* (LJ: URL);

– *de-* + *bug* «помилка» → *debug*: (63) *This allows specialists to implement / test / debug complex, task-specific objects, which you can then trust to run in your own code* (LJ: URL);

– *super-* + *class* «клас» → *superclass*: (65) *In the Java programming language, each class is allowed to have one direct superclass, and each superclass has the potential for an unlimited number of subclasses* (LJ: URL);

– *im-* + *mutable* «змінний» → *immutable*: (82) *String objects are immutable, which means that once created, their values cannot be changed* (LJ: URL);

б) **суфіксальний спосіб**:

– *compile* «збирати» + *-er* → *compiler*: (2) *This transformation is conducted by a special piece of software called a compiler* (LJ: URL);

– *initialize* «запускати, ініціалізувати» + *-er* → *initializer*: (90) *Since null has no type, the variable must have an initializer* (LJ: URL);

– *tutor* «вчитель» + *-ial* → *tutorial*: (4) *You will see how to download the JDK for free and how to install it later in this tutorial* (LJ: URL);

– *encapsulate* «укласти, наче в капсулу» + *-ion* → *encapsulation*: (21) *Multiple classes can be defined within the same source file if needed for encapsulation purposes, like in this example* (LJ: URL);

– *incubate* «інкубувати, виводити» + *-ing* → *incubating*: (42) *Incubating (also known as incubator modules), for potentially new APIs and JDK tools* (LJ: URL);

– *iterate* «повторити» + *-ion* → *iteration*: (49) *There might be one or multiple iterations of the preliminary access phase during which you have access to nonfinal features* (LJ: URL);

– *plug* «підключення» + *-ability* → *pluggability*: (64) *Pluggability and debugging ease: If a particular object turns out to be problematic, you can simply remove it from your application and plug in a different object as its replacement* (LJ: URL);

в) **префіксально-суфіксальний спосіб**:

– *pre-* + *load* «завантажений» + *-ed* → *preloaded*: (32) *Use this option to run only the scripts entered on the command line when JShell is started, or to start JShell without any preloaded information if no scripts are entered* (LJ: URL);

– *auto-* + *generate* «генерувати» + *-ed* → *auto-generated*: (78) *You may find some situations where auto-generated names will contain the dollar sign, but your variable names should always avoid using it* (LJ: URL);

– *self-* + *document* «документувати» + *-ing* → *self-documenting*: (79) *In many cases it will also make your code self-documenting; fields named cadence, speed, and gear, for example, are much more intuitive than abbreviated versions, such as s, c, and g* (LJ: URL);

– *un-* + *initialize* «почати, ініціалізувати» + *-ed* → *uninitialized*: (85) *Local variables are slightly different; the compiler never assigns a default value to an uninitialized local variable* (LJ: URL);

2) **конверсія** (перехід з однієї частини мови в іншу):

– *to build* «будувати» (дієслово) → *build* «збірка» (іменник): (10) *This page provides production-ready open-source builds of the Java Development Kit, an implementation of the Java SE Platform under the GNU General Public License, version 2, with the Classpath Exception* (LJ: URL);

– *literal* «буквенний» (прикметник) → *literal* «буквенний символ» (іменник): (88) *A literal is the source code representation of a fixed value; literals are represented directly in your code without requiring computation* (LJ: URL);

3) **словоскладання** (складання слів або основ слів в одне складне слово):

– *in* «такий, що знаходиться всередині» + *memory* «пам'ять» → *in-memory*: (20) *This works by the java launcher automatically invoking the compiler and storing the compiled code in-memory* (LJ: URL);

– *class* «клас» + *path* «шлях» → *classpath*: (22, 56) *A class that is part of the core JDK does not need to be added to the classpath to be executed* (LJ: URL);

– *over* «через край, повністю» + *write* «писати» → *overwrite*: (33) *Environment settings entered on the command line or provided with a previous /reset, /env, or /reload command are maintained unless an option is entered that overwrites the setting* (LJ: URL);

– *over* «через край, повністю» + *ride* «прямувати» → *override*: (34) *This option overrides the path in the CLASSPATH environment variable* (LJ: URL);

– *case* «регістр літер» + *sensitive* «чутливий» → *case-sensitive*: (77) *Variable names are case-sensitive* (LJ: URL);

– *statically* «статично» + *typed* «такий, що набирається на клавіатурі» → *statically-typed*: (81) *The Java programming language is statically-typed, which means that all variables must first be declared before they can be used* (LJ: URL);

– *compile* «компілювати, збирати» + *time* «час» → *compile-time*: (86) *Accessing an uninitialized local variable will result in a compile-time error* (LJ: URL);

4) **переосмислення** (використання загальновживаного слова в новому значенні):

– *to run* «бігти» → *to run* «запускати»: (1) *I understand that you are eager to type some code in your editor and run it to see your first Java application in action!* (LJ: URL);

– *class* «клас, тип об'єкту» → *class* «заявлений тип об'єкту в програмуванні»: (6) *If this technical term does not mean anything to you, do not worry, all you need to remember at this point is that all the code you write must be held in a Java class* (LJ: URL);

– *declaration* «оголошення, проголошення» → *declaration* «оголошення / визначення класів, змінних, тощо в програмуванні»: (7) *A Java class is created by a special declaration in a text file* (LJ: URL);

– *library* «бібліотека, місце, де зберігаються і доступні широкому загалу книги» → *library* «база даних відповідних значень»: *It is a set of tools and libraries to create enterprise-class applications* (LJ: URL);

– *script* «сценарій» → *script* «схема виконання програми»: (26) *To run a script after jshell is started, use the /open command* (LJ: URL);

– *snippet* «фрагмент» → *snippet* «сніппет»: (30) *The bits of code entered are called snippets* (LJ: URL);

– *bookmark* «закладка в книзі» → *bookmark* «закладка в браузері»: (72) *Load the page in your browser and bookmark it* (LJ: URL);

– *initialize* «стати ініціатором» → *initialize* «запустити програмний процес»: (73) *Do fields have to be initialized when they are declared?* (LJ: URL)

– *constructor* «будівельник, конструктор» → *constructor* «програма для розробки інших програм»: (89) *You can only use it for local variables declared in methods, constructors and initializer blocks* (LJ: URL);

– *operator* «оператор, управляючий» → *operator* «оператор, частина програмного коду»: (91) *Learning the operators of the Java programming language is a good place to start* (LJ: URL);

– *precedence* «перевага, першість» → *precedence* «пріоритет компонентів програмного коду»: (92) *When operators of equal precedence appear in the same expression, a rule must govern which is evaluated first* (LJ: URL);

5) **абревіація** (утворення лексичної одиниці на основі скорочення її компонентів):

– *input / output* «ввід / вивід» → *I/O*: (27) *To accept input from standard input and suppress the interactive I/O, enter a hyphen (-) for load-files* (LJ: URL);

– *identifier* «ідентифікатор» → *ID*: (38) *The ID for each snippet begins with the letter s, which indicates it's a startup snippet* (LJ: URL).

Професіоналізми у формі **словосполучень** представлені дво- та трикомпонентними словосполученнями. Зокрема, **двокомпонентні словосполучення** утворені за такими схемами творення:

1) **N + N** (іменник + іменник):

– *source* «джерело» + *code* «код» → *source code*: (3) *Whereas you can read a source code and understand it, binary or executable files are not meant to be read by a human person* (LJ: URL);

– *ZIP* (не абревіатура, оскільки походить від *zip* «блискавка (на одязі)» + file «файл»: (11) *What you get is a ZIP file of about 200MB that you can open with any ZIP utility software* (LJ: URL);

– *environment* «довкілля» + *variable* «змінна» → *environment variable*: (13) *Once this is done you need to create an environment variable called JAVA_HOME that points to the directory where you unzipped the JDK* (LJ: URL);

– *distribution* «розповсюдження» + *file* «файл» → *distribution file*: (15) *The exact directory depends on the distribution file you have expanded* (LJ: URL);

– *error* «помилка» + *message* «повідомлення» → *error message*: (16) *If it gives you an error message then you need to check your JAVA_HOME and PATH variables as there is most probably something wrong with them* (LJ: URL);

– *shebang* «усе, що стосується того, що розглядається» + *file* «файл» → *shebang file*: (23) *On a Unix-like operating system, a single-file source-code application, can also be launched as a shebang file like a script* (LJ: URL);

– *language* «мова» + *shell* «оболонка» → *language shell*: (24) *You use the language shell to learn the Java language, explore new features and APIs, and prototype new code* (LJ: URL);

– *startup* «дія або випадок приведення в дію чи рух» + *script* «сценарій» → *startup script*: (25) *Command-line scripts are run after startup scripts* (LJ: URL);

– *pipe* «труба» + *chain* «ланцюг»: (28) *This option enables the use of the jshell tool in pipe chains* (LJ: URL);

– *import* «введення» + *statement* «висловлювання» → *import statement*: (29) *Java statements, variable definitions, method definitions, class definitions, import statements, and expressions are accepted* (LJ: URL);

– *method* «метод» + *call* «виклик» → *method call*: (37) *When entering a method call, use the Tab key after the method call's opening parenthesis to see the parameters for the method* (LJ: URL);

– *scratch* «початок» + *variable* «змінна» → *scratch variable*: (39) *Note that a scratch variable is automatically created to hold the result because no variable was provided* (LJ: URL);

– *switch* «перемикання» + *expression* «вираз» → *switch expression*: (44) *Switch expressions is a language feature* (LJ: URL);

– *core* «ядро» + *platform* «платформа» → *core platform*: (47) *A Java SE API feature in the core Java platform, such as java.lang.Object, java.lang.String, or java.io.File* (LJ: URL);

– *name* «ім’я» + *space* «простір» → *namespace*: (55) *Finally, incubator modules are also shielded from accidental use because incubating can be done only in the jdk.incubator namespace* (LJ: URL);

– *data* «дані» + *encapsulation* «інкапсуляція» → *data encapsulation*: (60) *This section explains how state and behavior are represented within an object, introduces the concept of data encapsulation, and explains the benefits of designing your software in this manner* (LJ: URL);

– *code* «код» + *re-use* «повторне використання» → *code re-use*: (62) *Code re-use: If an object already exists (perhaps written by another software developer), you can use that object in your program* (LJ: URL);

– *class* «клас» + *declaration* «оголошення» → *class declaration*: (66) *At the beginning of your class declaration, use the extends keyword, followed by the name of the class to inherit from* (LJ: URL);

– *class* «клас» + *library* «бібліотека» → *class library*: (68) *The Java platform provides an enormous class library (a set of packages) suitable for use in your own applications* (LJ: URL);

– *network* «мережа» + *socket* «розетка» → *network socket*: (70) *A Socket object allows for the creation and use of network sockets* (LJ: URL);

– *exception* «виняток» + *handler* «обробник» → *exception handler*: (76) *This applies to other parameter-accepting constructs as well (such as constructors and exception handlers) that you'll learn about later in the tutorial* (LJ: URL);

– *assignment* «призначення, присвоєння» + *operator* «оператор» → *assignment operator*: (93) *In general-purpose programming, certain operators tend to appear more frequently than others; for example, the assignment operator = is far more common than the unsigned right shift operator >>>* (LJ: URL);

– *object* «об'єкт» + *reference* «посилання» → *object reference*: (94) *This operator can also be used on objects to assign object references, as discussed in the section Creating Objects* (LJ: URL);

– *method* «метод» + *invocation* «виклик» → *method* invocation: (98) *An expression is a construct made up of variables, operators, and method invocations, which are constructed according to the syntax of the language, that evaluates to a single value* (LJ: URL);

2) **Abbr + N** (абревіатура + іменник), наприклад, *DOS (Disk Operating System)* «вбудована операційна система комп'ютера» + *prompt* «підказка» → *DOS prompt*: (14) *First you need to open a DOS prompt* (LJ: URL);

3) **Adj + N** (прикметник + іменник):

– *executable* «виконуваний» + *code* «код» → *executable code*: (17) *So far your class is empty; there is no executable code in it* (LJ: URL);

– *verbose* «багатослівний» + *mode* «режим» → *verbose mode*: (31) *Start with the verbose mode to get the most feedback while learning the tool* (LJ: URL);

– *critical* «критичний» + *workload* «робоче навантаження» → *critical workload*: (40) *With millions of people relying on Java in production for critical workloads, the reach of Java is global* (LJ: URL);

– *preliminary* «попередній» + *access* «доступ» → *preliminary access*: (41) *It is therefore critical to provide developers with preliminary access to new features to encourage feedback – feedback that will help refine those features and reach the expected quality level for their final and permanent form* (LJ: URL);

– *nonfinal* «нефінальний, неостаточний» + *feature* «властивість, функція» → *nonfinal feature*: (43) *In addition, there are other nonfinal features that don't fit in any of those three categories, that you will see later* (LJ: URL);

– *early* «ранній» + *access* «доступ» → *early access*: (58) *The unique goal of such occasional feature-specific early access (EA) JDK builds is solely to allow expert users to test specific novel features early* (LJ: URL);

– *non-static* «нестатичний, рухомий» + *field* «поле» → *non-static field*: (74) *Non-static fields are also known as instance variables because their values are unique to each instance of a class (to each object, in other words)* (LJ: URL);

– *local* «місцевий» + *variable* «змінна» → *local variable*: (75) *Similar to how an object stores its state in fields, a method will often store its temporary state in local variables* (LJ: URL);

– *reasonable* «розумний» + *default* «значення за замовчуванням» → *reasonable default*: (83) *Fields that are declared but not initialized will be set to a reasonable default by the compiler* (LJ: URL);

– *primitive* «примітивний» + *type* «тип» → *primitive type*: (87) *Primitive types are special data types built into the language; they are not objects created from a class* (LJ: URL);

– *unary* «одинарний» + *operator* «оператор» → *unary operator*: (95) *The unary operators require only one operand; they perform various operations such as incrementing / decrementing a value by one, negating an expression, or inverting the value of a boolean* (LJ: URL);

– *ternary* «трійковий» + *operator* «оператор» → *ternary operator*: (96) *This operator is also known as the ternary operator because it uses three operands* (LJ: URL);

Окремо в цій групі представлені словосполучення, утворені за схемою Adj + N (складне слово у ролі прикметника + іменник):

– (*test bed* «випробувальний стенд» → *test-bed*) + *mechanism* «механізм» → *test-bed mechanism*: (50) *An experimental feature is a test-bed mechanism used to gather feedback on nontrivial HotSpot enhancements* (LJ: URL);

– (*backward compatibility* «зворотна сумісність» → *backward-compatibility*) + *rule* «правило» → *backward-compatibility rule:* (51) *Moreover, only final, permanent features are subject to Java's stringent backward-compatibility rules* (LJ: URL);

– (*object* «об'єкт» + *oriented* «орієнтований» → *object-oriented*) + *programming* «програмування» → *object-oriented programming*: (61) *This section explains how state and behavior are represented within an object, introduces the concept of data encapsulation, and explains the benefits of designing your software in this manner* (LJ: URL);

– (*general* «загальний» + *purpose* «мета» → *general-purpose*) + *programming* «програмування» → *general-purpose programming*: (69) *Its packages represent the tasks most commonly associated with general-purpose programming* (LJ: URL);

4) **V + N** (дієслово + іменник):

– *set* «встановити» + *(to)* + *default* «значення за замовчуванням» → *set to default*: (35) *Sets the editor to the default editor provided with JShell. This option can't be used if a command for starting an editor is entered* (LJ: URL);

– *compile* «компілювати» + *time* «час» → *compile time*: (53) *Preview features are specific to a given Java SE feature release and require the use of special flags at compile time as well as at runtime* (LJ: URL);

– *build* «будувати» + *time* «час» → *build time*: (67) *Interfaces form a contract between the class and the outside world, and this contract is enforced at build time by the compiler* (LJ: URL);

5) **PI + N** (дієприкметник Participle I + іменник), наприклад, *incubating* «інкубуючий, той, що виводить» + *feature* «властивість, функція» → *incubating feature*: (46) *The HTTP/2 Client API is an API added as an incubating feature in Java SE 9 and Java SE 10, and was then made a standard feature in Java SE 11* (LJ: URL);

6) **PII + N** (дієприкметник Participle II + іменник), наприклад, *reserved* «зарезервований» + *word* «слово» → *reserved word*: (80) *Also keep in mind that the name you choose must not be a keyword or reserved word* (LJ: URL).

Професіоналізми у формі **трикомпонентних словосполучень** творяться за такими схемами:

1) **N + N + N** ([іменник + іменник] + іменник, де перші два іменника позначають готовий професіоналізм, а новий професіоналізм твориться із додаванням нового компонента до двокомпонентного), наприклад, [*source* «джерело» + *code* «код»] + *repository* «вмістилище, сховище» → *source code repository*: (48) *If there is a high demand for an enhancement, if it impacts the JDK*

or the processes and infrastructure by which the JDK itself is developed (such as moving the JDK source code repository to GitHub), or simply because it requires quite a bit of engineering investment, then it is nontrivial (LJ: URL);

2) **Adj + N + N** представлено таким випадками:

а) Adj + N + N ([прикметник + іменник] + іменник, де перші прикметника утворюють двокомпонентний професіоналізм, додаванням до іменника до якого твориться новий професіоналізм):

– [*plain* «простий» + *text* «текст»] + *file* «файл» → *plain text file*: *The first step you need to know is that the Java code you are writing is saved in plain text files* (LJ: URL);

– [*floating* «плавачий» + *point* «крапка»] + *arithmetic* «арифметика» → *floating point arithmetic*: (99) *Floating point arithmetic is a special world in which common operations may behave unexpectedly* (LJ: URL).

У цій групі також спостерігається випадок творення Adj + N + N ([складне слово у ролі прикметника + іменник] + іменник), наприклад: (*object* «об'єкт» + *oriented* «орієнтований» → *object-oriented*) + [*programming* «програмування» + *language* «мова»] → *object-oriented programming language*: (59) *If you've never used an object-oriented programming language before, you will need to learn a few basic concepts before you can begin writing any code* (LJ: URL);

б) Adj + N + N (прикметник + [іменник + іменник], де прикметник додається до готового професіоналізма, що складається з двох іменників):

– *graphical* «графічний» + [*user* «користувач» + *interface* «інтерфейс»] → *graphical user interface*: (71) *Various GUI objects control buttons and check boxes and anything else related to graphical user interfaces* (LJ: URL);

– *bad* + [*programming* «програмування» + *style* «стиль»] → *bad programming style*: (84) *Relying on such default values, however, is generally considered bad programming style* (LJ: URL);

3) **PII + N + N** (дієприкметник Participle II + [іменник + іменник], де дієприкметник додається до готового іменникового професіоналізма),

наприклад, *integrated* «інтегрований» + [*development* «розвиток, розробка» + *environment* «середовище»] → *integrated development environment*: (18) *Developers that work on large applications do not use plain text editors to manage their source code; they use Integrated Development Environments* (LJ: URL);

4) **Letter + N + N** (літера як окремий компонент + іменник + іменник), наприклад, *Z* (літера не має фактичного значення, обиралася розробниками за милозвучністю загальної абревіатури *ZGC* на основі назви *ZFS* від Oracle (SO: URL)) + *garbage* «сміття» + *collector* «збірник» → *Z garbage collector*: (45) *The Z garbage collector is a HotSpot feature* (LJ: URL).

Кількісний аналіз способів творення професіоналізмів IT дискурсу сфери програмування представлено у *Таблиці 2.2*.

Таблиця 2.2

Способи творення професіоналізмів у контексті програмування IT дискурсу

Способи творення	**Кількість**	**Частка**
1. Слова	**42**	**42%**
афіксація	18	18%
- префіксація	7	7%

Закінчення Таблиці 2.2

Способи творення	**Кількість**	**Частка**
- суфіксація	7	7%
- префіксально-суфіксальний спосіб	4	4%
конверсія	2	2%
словоскладання	7	7%
переосмислення	13	13%
абревіація	2	2%

2. Словосполучення	**58**	**58%**
а) двокомпонентні	50	50%
- N + N	28	28%
- Abbr + N	1	1%
- Adj + N	16	16%
- V + N	3	3%
- PI + N	1	1%
- PII + N	1	1%
б) трикомпонентні	8	8%
- N + N + N	1	1%
- Adj + N + N	5	5%
- PII + N + N	1	1%
- Letter + N + N	1	1%
Загалом	**100**	**100%**

Таким чином, проведене дослідження демонструє, що професіоналізми у контексті програмування IT дискурсу практично і рівній мірі представлені словами (42%) та словосполученнями (58%) із кількісною перевагою словосполучень. Серед словосполучень найчастіше професіоналізми в контексті програмування IT дискурсу представлені двокомпонентними словосполученнями (50%). Найбільш продуктивними способами творення професіоналізмів у контексті програмування IT дискурсу у формі слів постають афіксація (18%), переосмислення (13%) та словоскладання (7%).

Висновки до розділу 2

1. Професіоналізми у контексті програмування IT дискурсу розподілено на тематичні групи і підгрупи: 1) група «людина» (підгрупи «учасники процесу програмування», «характеристики учасників процесу

програмування»); 2) група «процес програмування» (підгрупи «мова програмування» [підгрупи другого рівня «мова програмування взагалі», «складові мови програмування» {підгрупи третього рівня «класи», «дані», «бібліотеки», «змінні», «вирази», «команди»}], «програмний код та його складові» [підгрупи другого рівня «програмний код», «складові програмного коду», «функції»]; «стадії програмування», «складові процесу програмування» [підгрупи другого рівня «програмне забезпечення для програмування», «сценарії виконання програм», «характеристики програмованого продукту»], «дії в процесі програмування» [підгрупи другого рівня «дії», «оцінка ефективності дій»]); 3) група «програмне забезпечення» («робота програмного забезпечення» [підгрупи другого рівня «засоби взаємодії», «рівні взаємодії», «способи взаємодії»], «взаємодія програми з користувачем», «файлова система та типи файлів»); 4) група «апаратне забезпечення» («обчислювальне обладнання», «мережеве обладнання»); 5) група «суміжні науки». Професіоналізми у контексті програмування ІТ дискурсу найчастіше стосуються процесу програмування (72%), зокрема, дій у процесі програмування (16%) та складових процесу програмування (13%). 20% професіоналізмів у контексті програмування ІТ дискурсу стосується програмного забезпечення, особливо – взаємодії програми з користувачем (11%). Найменш частотними тематичними групами професіоналізмів у контексті програмування ІТ дискурсу постають професіоналізм на позначення апаратного забезпечення (4%), людини як учасника процесу програмування (2%) та професіоналізми суміжних наук (2%).

2. Професіоналізми в контексті програмування ІТ дискурсу творяться за різними моделями, як результат, вони представлені практично в рівній мірі представлені словами (42%) та словосполученнями (58%) із кількісною перевагою словосполучень. Зокрема, серед словосполучень (58%) найчастіше професіоналізми в контексті програмування ІТ дискурсу представлені двокомпонентними словосполученнями (50%), переважно – утвореними за

схемами N + N (28%) та Adj + N (16%). Серед трикомпонентних словосполучень (8%) професіоналізми в контексті програмування IT дискурсу найчастіше творяться за схемою Adj + N + N (5%). Найбільш продуктивними способами творення професіоналізмів у контексті програмування IT дискурсу у формі слів (42%) постають афіксація (18%, переважно – префіксація та суфіксація (по 7% на групу)), переосмислення (13%) та словоскладання (7%).

РОЗДІЛ 3
ОСОБЛИВОСТІ ВІДТВОРЕННЯ В УКРАЇНСЬКОМУ ПЕРЕКЛАДІ ПРОФЕСІОНАЛІЗМІВ У КОНТЕКСТІ ПРОГРАМУВАННЯ ІТ ДИСКУРСУ

3.1 Лексичні перекладацькі трансформації при перекладі професіоналізмів у контексті програмування ІТ дискурсу

Особливості комунікації у дискурсі інформаційних технологій, а особливо – в контексті програмування, де значна кількість інформації початково створюється англійською мовою, після чого запозичується до різних мов, зумовлюють використання при відтворенні в україномовних перекладах професіоналізмів у контексту програмування ІТ дискурсу шляхом застосування лексичних перекладацьких трансформацій, до яких належать нульовий переклад, практична транскрипція, транслітерація та калькування.

Нульовий переклад – це відтворення професіоналізму зі збереженням його вихідної форми. Застосування нульового перекладу зумовлено тим, що українці все частіше сприймають на письмі окремі унікальні для програмування назви саме у вихідній формі. У цей спосіб відтворено такі професіоналізми у контексті програмування ІТ дискурсу:

– *switch expression – "switch"-вирази*, оскільки *switch* – це назва оператора мови програмування, який використовується у програмному коді саме в такому вигляді, тому доцільно і при перекладі зберегти вихідну форму назви оператора: (44) *Switch expressions is a language feature* (LJ: URL) – "Switch"-вирази – це функція мови;

– *Z garbage collector – збірник сміття Z*, де *Z* – літера, що не має конкретного смислового навантаження, а лише відрізняє одну версію програмного забезпечення від іншої, тому при перекладі вона зберігається у

своїй вихідній формі: (45) *The Z garbage collector is a HotSpot feature* (LJ: URL) – Збірник сміття Z – це функція HotSpot.

Наступний спосіб передачі при перекладі професіоналізмів у контексті програмування ІТ дискурсу – **практична транскрипція**, що передбачає передачу звукової форми професіоналізму засобами алфавіту української мови.

Найпростішим способом використання практичної транскрипції є повне транскрибування лексичної одиниці, як у наступних фрагментах:

– *script – скрипт*: (26) *To run a script after jshell is started, use the /open command* (LJ: URL) – Щоб запустити скрипт після запуску jshell, скористайтеся командою /open;

– *interface – інтерфейс*: (54) *You need a first mandatory Square class, which you need to store in the same package or module as the Shape interface* (LJ: URL) – Вам потрібен перший обов'язковий клас Square, який потрібно зберігати в тому ж пакеті або модулі, що й інтерфейс Shape.

Практичній транскрипції може підлягати не лише лексична одиниця повністю, а і її окремі компоненти, наприклад, *scratch variable – змінна скретчу*: (39) *Note that a scratch variable is automatically created to hold the result because no variable was provided* (LJ: URL) – Зауважте, що змінна скретчу створюється автоматично для збереження результату, оскільки змінна не була надана.

При застосуванні практичної транскрипції також можуть відбуватися певні зміни в буквенному наповненні слова в перекладі, що пов'язано з особливостями фонетики мови перекладу, наприклад, *class – клас*: (6) *If this technical term does not mean anything to you, do not worry, all you need to remember at this point is that all the code you write must be held in a Java class* (LJ: URL) – Якщо цей технічний термін для вас нічого не означає, не хвилюйтеся, все, що вам потрібно пам'ятати, – це те, що весь код, який ви пишете, повинен зберігатися в класі Java.

Окрім того, може відбуватися і більш суттєва адаптація до норм мови перекладу з додаванням суфіксів та закінчень, типових для мови перекладу та відсутніх у лексичній одиниці мовою оригіналу:

– *distribution file – файл дистрибутиву*, де *distribution* передано як *дистрибутив*, тобто, із зміною суфіксу лексичної одиниці: (15) *The exact directory depends on the distribution file you have expanded* (LJ: URL) – Точний каталог залежить від файлу дистрибутиву, який ви розгорнули;

– *encapsulation – інкапсуляція*, де відбувається адаптація суфіксу та додавання родового закінчення відповідно до норм української мови: (21) *Multiple classes can be defined within the same source file if needed for encapsulation purposes, like in this example* (LJ: URL) – Декілька класів можна визначити в одному вихідному файлі, якщо це необхідно для цілей інкапсуляції, як у цьому прикладі.

Довга традиція комунікації між програмістами різних країн у письмовому вигляді призводить до того, що значна кількість професіоналізмів у контексті програмування ІТ дискурсу відтворюється при перекладі українською мовою із застосуванням **транслітерації** – запозичення графічної форми професіоналізму.

Зокрема, при відтворенні професіоналізмів у контексті програмування ІТ дискурсу спостерігаються чисельні випадки застосування транслітерації в чистому вигляді, наприклад:

– *tutorial – туторіал*: (4) *You will see how to download the JDK for free and how to install it later in this tutorial* (LJ: URL) – Ви дізнаєтеся, як безкоштовно завантажити JDK і як його встановити, пізніше в цьому туторіалі;

– *snippet – сніппет*: (30) *The bits of code entered are called snippets* (LJ: URL) – Введені біти коду називаються сніппетами;

– *literal – літерал*: (88) *A literal is the source code representation of a fixed value; literals are represented directly in your code without requiring computation*

(LJ: URL) – Літералом є вихідне представлення фіксованого значення; літерали представлені безпосередньо у вашому коді, не вимагаючи обчислень;

– *constructor – конструктор*: (89) *You can only use it for local variables declared in methods, constructors and initializer blocks* (LJ: URL) – Ви можете використовувати його лише для локальних змінних, оголошених у методах, конструкторах та блоках ініціалізації;

– *operator – оператор*: (91) *Learning the operators of the Java programming language is a good place to start* (LJ: URL) – Вивчення операторів мови програмування Java – гарне місце для початку.

Як і у випадку практичної транскрипції, транслітерація може застосовуватися і до окремого компоненту професіоналізму, в той час, як інший компонент відтворюється при перекладі іншим способом:

– *network socket – мережевий сокет:* (70) *A Socket object allows for the creation and use of network sockets* (LJ: URL) – Об'єкт Socket дозволяє створювати та використовувати мережеві сокети;

– *shebang file – файл шебанг:* (23) *On a Unix-like operating system, a single-file source-code application, can also be launched as a shebang file like a script* (LJ: URL) – У Unix-подібній операційній системі однофайлова програма з вихідним кодом також може бути запущена файлом шебанг як сценарій.

Застосування транслітерації також може вимагати адаптації транслітерованої основи до норм мови перекладу, зокрема, додавання суфіксів та закінчень:

– *compiler – компілятор:* (2) *This transformation is conducted by a special piece of software called a compiler* (LJ: URL) – Це перетворення виконується спеціальним програмним забезпеченням, яке називається «компілятор»;

– *incubating – інкубаційний*: (42) *Incubating (also known as incubator modules), for potentially new APIs and JDK tools* (LJ: URL) – Інкубаційні (також відомі як інкубаційні модулі) для потенційно нових API та інструментів JDK;

– *iteration – ітерація*: (49) *There might be one or multiple iterations of the preliminary access phase during which you have access to nonfinal features* (LJ: URL) – Може бути одна або кілька ітерацій фази попереднього доступу, під час якої ви маєте доступ до нефінальних функцій;

– *unary operator – унарний оператор*, де *unary* відтворено як *унарний*: (95) *The unary operators require only one operand; they perform various operations such as incrementing / decrementing a value by one, negating an expression, or inverting the value of a boolean* (LJ: URL) – Унарні оператори вимагають лише однієї операнди; вони виконують різні операції, такі як збільшення / зменшення значення на одиницю, заперечення виразу або інвертування логічного значення.

Перекладацька трансформація **калькування** передбачає поелементне відтворення багатокомпонентних професіоналізмів у контексті програмування ІТ дискурсу, компоненти яких мають усталені еквіваленти у мові перекладу.

Зокрема, калькування є ефективним при відтворенні професіоналізмів у формі слів, утворених префіксальним (або префіксально-суфіксальним) способом або шляхом словоскладання, коли кожен компонент такої одиниці перекладається окремо, після чого з отриманих компонентів у перекладі утворюється українськомовний професіоналізм:

– *preloaded – попередньо завантажений*, де префікс *pre-* має відповідник *попередньо*, а *loaded* відтворено прикметником *завантажений*: (32) *Use this option to run only the scripts entered on the command line when JShell is started, or to start JShell without any preloaded information if no scripts are entered* (LJ: URL) – Використовуйте цей параметр, щоб запускати лише сценарії, введені в командному рядку під час запуску JShell, або запускати JShell без попередньо завантаженої інформації, якщо сценарії не введені;

– *overwrite – перезаписати*, де *over-* передано як *пере-*, а *write* – відповідником *записати*: (33) *Environment settings entered on the command line or provided with a previous /reset, /env, or /reload command are maintained unless*

an option is entered that overwrites the setting (LJ: URL) – Параметри середовища, введені в командному рядку або надані попередньою командою /reset, /env або /reload, зберігаються, якщо не введено параметр, який перезаписує налаштування;

– *subcommand – підкоманда*, де префікс *sub-* передано префіксом *під-*, а *command* відтворено прямим відповідником *команда*: (36) *When entering snippets, commands, subcommands, command arguments, or command options, use the Tab key to automatically complete the item* (LJ: URL) – Під час введення сніппетів, команд, підкоманд, аргументів команди або параметрів команди використовуйте клавішу Tab, щоб автоматично завершити елемент;

– *immutable – незмінний*, де заперечний префікс *im-* передано відповідним йому українськомовним префіксом *не-*, а *mutable* передано українським відповідником *змінний*: (82) *String objects are immutable, which means that once created, their values cannot be changed* (LJ: URL) – Рядкові об’єкти незмінні, що означає, що після створення їх значення не можна змінити.

Застосування калькування також доцільне при передачі значної кількості професіоналізмів у формі двокомпонентних словосполучень, таких як:

– *executable code – виконуваний код*: (17) *So far your class is empty; there is no executable code in it* (LJ: URL) – Поки що ваш клас порожній; в ньому немає виконуваного коду;

– *critical workload – критичне робоче навантаження*: (40) *With millions of people relying on Java in production for critical workloads, the reach of Java is global* (LJ: URL) – Оскільки мільйони людей покладаються на Java у створенні програмного забезпечення, що могло б витримувати критичні робочі навантаження, використання Java є глобальним;

– *preliminary access – попередній доступ*: (41) *It is therefore critical to provide developers with preliminary access to new features to encourage feedback –*

feedback that will help refine those features and reach the expected quality level for their final and permanent form (LJ: URL) – Тому дуже важливо надати розробникам попередній доступ до нових функцій, щоб заохочувати зворотний зв'язок – зворотний зв'язок, який допоможе вдосконалити ці функції та досягти очікуваного рівня якості для їх остаточної та постійної форми;

– *early access – ранній доступ*: (58) *The unique goal of such occasional feature-specific early access (EA) JDK builds is solely to allow expert users to test specific novel features early* (LJ: URL) – Унікальна мета таких випадкових збірок JDK для специфічного раннього доступу (early access – EA) полягає виключно в тому, щоб дозволити експертним користувачам завчасно тестувати конкретні нові функції;

– *non-static field – нестатичне поле*: (74) *Non-static fields are also known as instance variables because their values are unique to each instance of a class (to each object, in other words)* (LJ: URL) – Нестатичні поля також відомі як змінні екземпляра, оскільки їх значення є унікальними для кожного екземпляра класу (іншими словами, для кожного об'єкта);

– *unit of execution – одиниця виконання*: (100) *A statement forms a complete unit of execution* (LJ: URL) – Оператор утворює повну одиницю виконання.

Зміна порядку слів у таких словосполученнях викликана не бажанням перекладача, а особливостями використання іменникових словосполучень в англійській та українській мовах:

– *environment variable – змінна середовища:* (13) *Once this is done you need to create an environment variable called JAVA_HOME that points to the directory where you unzipped the JDK* (LJ: URL) – Після цього вам потрібно створити змінну середовища під назвою JAVA_HOME, яка вказує на каталог, де ви розпакували JDK;

– *startup script – сценарій запуску*: (25) *Command-line scripts are run after startup scripts* (LJ: URL) – Скрипти командного рядка запускаються після сценаріїв запуску;

– *method call – виклик методу*: (37) *When entering a method call, use the Tab key after the method call's opening parenthesis to see the parameters for the method* (LJ: URL) – Під час введення виклику методу використовуйте клавішу Tab після відкриваючої дужки виклику методу, щоб побачити параметри методу;

– *build time – час збірки*: (67) *Interfaces form a contract between the class and the outside world, and this contract is enforced at build time by the compiler* (LJ: URL) – Інтерфейси утворюють узгодження між класом і зовнішнім світом, і це узгодження виконується компілятором під час збірки.

Калькування також є доцільним при перекладі трикомпонентних професіоналізмів, таких як *integrated development environment – інтегроване середовище розробки:* (18) *Developers that work on large applications do not use plain text editors to manage their source code; they use Integrated Development Environments* (LJ: URL) – Розробники, які працюють над великими програмами, не використовують звичайні текстові редактори для керування вихідним кодом; вони використовують інтегровані середовища розробки.

Окрім того, калькування може використовуватися при покомпонентній передачі абревіатур, коли процедура калькування застосовується до повних назв, наприклад, *I/O (input / output) – ввід / вивід*. У таких випадках у перекладі застосовується повна, а не скорочена форма професіоналізму: (27) *To accept input from standard input and suppress the interactive I/O, enter a hyphen (-) for load-files* (LJ: URL) – Щоб прийняти стандартне введення та придушити інтерактивний ввід-вивід, введіть дефіс (-) для завантажуваних файлів.

Таким чином, під час відтворення англійськомовних професіоналізмів в контексті програмування ІТ дискурсу українською мовою такі одиниці можуть залишатися у вихідній формі, коли вони є елементами коду, що не

перекладається при використанні, або ж власних назв, або ж переклад може здійснюватися з використанням транскрипції і транслітерації, коли відтворюються загальновідомі у світі професіоналізми, або калькування, коли компоненти професіоналізмів мають усталені відповідники українською мовою.

3.2 Лексико-семантичні перекладацькі трансформації як засіб передачі професіоналізмів при перекладі у контексті програмування ІТ дискурсу

Оскільки при сприйманні професійного тексту розуміння значення кожного із його складників є надзвичайно важливим, у деяких випадках при відтворенні українською мовою професіоналізмів у контексті програмування ІТ дискурсу використовуються лексико-семантичні перекладацькі трансформації, що передбачають модифікацію значення одиниці, що перекладається. До таких перекладацьких трансформацій належать диференціація, генералізація, конкретизація та модуляція.

Диференціація значення викликана тим, що велика кількість англійських слів з широкою семантикою не має повних еквівалентів в українській мові. У таких випадках у словниках представлена певна кількість значень які лише частково покривають значення слова мови оригіналу, а перекладач обирає такий варіант, який найкраще підходить до певного контексту [42: 114].

Наприклад, акт визначення змінних у програмному коді в мовах програмування іменується *declaration* ‘проголошення’, ‘заява’, ‘оголошення’, що у перекладі відтворюється як *оголошення*, тобто, значення слова у перекладі обирається з урахуванням ситуації комунікації: (7) *A Java class is created by a special declaration in a text file* (LJ: URL) – Клас Java створюється за допомогою спеціального «оголошення» в текстовому файлі.

Наступний приклад – *library* ‘бібліотека’, ‘книгосховище’, і, оскільки в контексті програмування йдеться саме про сховище даних, то більш доцільним варіантом постає *бібліотека*, що зберігає ознаку переосмислення загальновживаної мовної одиниці: (9) *It is a set of tools and libraries to create enterprise-class applications* (LJ: URL) – Це набір інструментів і бібліотек для створення додатків корпоративного класу.

У словосполучення *nonfinal feature* іменник *feature* ‘функція’, ‘особливість’, ‘характеристика’ передано як *функція*, оскільки йдеться про певний набір правил, що реалізується в мові програмування: (43) *In addition, there are other nonfinal features that don’t fit in any of those three categories, that you will see later* (LJ: URL) – Крім того, є інші нефінальні функції, що не вписуються в жодну з цих трьох категорій, і їх ви побачите пізніше.

Диференціація застосовується також при відтворенні професіоналізму *source code repository*, де *source* ‘джерело’, ‘походження’ передано як *вихідний* (також спостерігається заміна іменника прикметником, граматичні заміни висвітлено у п. 3.3): (48) *If there is a high demand for an enhancement, if it impacts the JDK or the processes and infrastructure by which the JDK itself is developed (such as moving the JDK source code repository to GitHub), or simply because it requires quite a bit of engineering investment, then it is nontrivial* (LJ: URL) – Якщо є високий попит на покращення, якщо це впливає на JDK або процеси та інфраструктуру, за допомогою яких розробляється сам JDK (наприклад, переміщення сховища вихідного коду JDK на GitHub), або просто тому, що це вимагає значних інвестицій в інженерію, то це нетривіально.

Іменник *precedence* ‘перевага’, ‘передування’, ‘першість’ у контексті програмування як показник ієрархії операторів передається як *пріоритет*: (92) *When operators of equal precedence appear in the same expression, a rule must govern which is evaluated first* (LJ: URL) – Коли в одному виразі з’являються оператори однакового пріоритету, повинно бути правило, що визначає, який оператор використовується першим.

У словосполученні *method invocation* іменник *invocation* ‘звернення’, ‘виклик’ передається як *виклик*, оскільки описувані компоненти програмного коду використовуються саме для активації певних методів мови програмування: (98) *An expression is a construct made up of variables, operators, and method invocations, which are constructed according to the syntax of the language, that evaluates to a single value* (LJ: URL) – Вираз – це конструкція, що складається зі змінних, операторів і викликів методів, які створюються відповідно до синтаксису мови, що обчислюється до одного значення.

Іменник *handler* ‘обробник’, ‘тренер’ у словосполученні *exception handler* застосовується для позначення компоненту програми, яка використовується для вирішення проблемних ситуацій, пов’язаних із винятками, тому в цьому випадку найбільш доцільним варіантом перекладу постає *обробник*: (76) *This applies to other parameter-accepting constructs as well (such as constructors and exception handlers) that you’ll learn about later in the tutorial* (LJ: URL) – Це також стосується інших конструкцій, що приймають параметри (таких як конструктори та обробники винятків), про які ви дізнаєтеся пізніше.

У деяких випадках диференціації підлягають і окремі компоненти лексичних одиниць, утворених шляхом афіксації або словоскладання, наприклад:

– *superclass* – *надклас*, де *super-* ‘супер-’, ‘най-’, ‘над-’ відтворено як *над-*, оскільки йдеться про ієрархії класів у мові програмування: (65) *In the Java programming language, each class is allowed to have one direct superclass, and each superclass has the potential for an unlimited number of subclasses* (LJ: URL) – У мові програмування Java кожному класу дозволено мати один прямий надклас, і кожен надклас може мати необмежену кількість підкласів;

– *general-purpose programming*, де *purpose* ‘мета’, ‘намір’, ‘призначення’ передано як *призначення*, оскільки йдеться саме про призначення мови програмування: (69) *Its packages represent the tasks most*

commonly associated with general-purpose programming (LJ: URL) – Його пакети представляють завдання, які найчастіше пов'язані з програмуванням загального призначення.

Прийом **генералізації** полягає у заміні одиниці мови оригіналу. що має більш вузьке значення, одиницею мови перекладу з більш широким значенням. Застосування цього прийому може бути вимушеним або факультативним. У першому випадку в мові перекладу відсутнє необхідне слово з конкретним значенням. У другому випадку перекладач віддає перевагу більш загальному варіанту зі стилістичних міркувань [42: 113].

Генералізація може застосовуватися у випадках, коли можливий лише приблизний переклад професіоналізму. Так, наприклад, *revamp* передбачає не просто оновлення програми, а її «перезалив», тобто, заміну однієї версії іншою повністю. Оскільки матеріалом дослідження є електронний підручник, розрахований на досить широку аудиторію, то в перекладі це значення передано більш загально дієсловом *оновити*: (57) *The goal of those projects is to conduct fundamental investigations in particular areas to drastically improve (or completely revamp) certain aspects of the Java platform* (LJ: URL) – Метою цих проектів є проведення фундаментальних досліджень у певних областях, щоб радикально покращити (або повністю оновити) певні аспекти платформи Java.

Генералізація також може бути викликана усталеними способами передачі тих чи інших значень в українській мові, тобто, випадками, коли у спільноті програмувальників уже є усталені варіанти перекладу тих чи інших явищ більш загальними словами, наприклад:

– *core platform – основна платформа* (*core* ‘серединний’, ‘ядерний’ → основний): (47) *A Java SE API feature in the core Java platform, such as java.lang.Object, java.lang.String, or java.io.File* (LJ: URL) – Функція API Java SE в основній платформі Java, наприклад java.lang.Object, java.lang.String або java.io.File;

– *unzip* ‘розархівувати’ – *розпакувати*: (12) *You can unzip the content of this file anywhere on your computer* (LJ: URL) – Ви можете розпакувати вміст цього файлу будь-де на своєму комп’ютері;

– *override* ‘перевизначити’ – *замінити*: (34) *This option overrides the path in the CLASSPATH environment variable* (LJ: URL) – Цей параметр замінює шлях у змінній середовища CLASSPATH.

Прийом **конкретизації** полягає в тому, що перекладач вибирає для перекладу слова оригіналу слово з більш конкретним значенням у мові перекладу [42: 113].

Конкретизація дозволяє уточнити значення певних мовних одиниць у перекладі, що дозволяє уникнути плутанини при розумінні тексту. Наприклад, *import statement*, де *statement* ‘судження’, ‘констатування’ замінено на *оператор*, що вказує на природу позначуваного явища – *оператор імпорту*, тобто, команда, яка виконується комп’ютером: (29) *Java statements, variable definitions, method definitions, class definitions, import statements, and expressions are accepted* (LJ: URL) – Приймаються оператори Java, визначення змінних, визначення методів, визначення класів, оператори імпорту та вирази.

У вислові *set to the default* компонент *to the default* ‘значення за замовчуванням’ передано як *стандартні значення* з огляду на особливості вживання такої термінології у спілкуванні англомовних та україномовних програмістів: (35) *Sets the editor to the default editor provided with JShell. This option can't be used if a command for starting an editor is entered* (LJ: URL) – Встановлює для редактора стандартні налаштування, надані JShell. Цей параметр не можна використовувати, якщо введена команда для запуску редактора.

Ще один приклад уточнення – назва оператору *assignment operator*, де *assignment* ‘завдання’, ‘призначення’ передається як *присвоєння – оператор присвоєння* – що дозволяє представити сутність цього оператора через його назву: (93) *In general-purpose programming, certain operators tend to appear*

more frequently than others; for example, the assignment operator = is far more common than the unsigned right shift operator >>> (LJ: URL) – У програмуванні загального призначення деякі оператори мають тенденцію з'являтися частіше, ніж інші; наприклад, оператор присвоєння = набагато частіше зустрічається, ніж беззнаковий оператор зсуву вправо >>>.

Модуляцією, або смисловим розвитком, називається заміна слова або словосполучення мови оригіналу одиницею мови перекладу, значення якої логічно виводиться із значення вихідної одиниці. Найбільш часто значення співвіднесених відрізків в оригіналі і перекладі опиняються при цьому зв'язаними причинно-наслідковими відносинами. І в цьому випадку відмова від «прямого» перекладу може бути вимушеною або залежати від вибору перекладача [42: 114].

Зокрема, причинно-наслідкові відносини процесу і результату спостерігаються при відтворенні *build* 'будівля', 'конструкція' як *збірка*, тобто, ідея про побудову чогось репрезентована ідеєю про збирання чогось воєдино: (10) *This page provides production-ready open-source builds of the Java Development Kit, an implementation of the Java SE Platform under the GNU General Public License, version 2, with the Classpath Exception* (LJ: URL) – На цій сторінці представлені готові до використання збірки Java Development Kit, реалізація платформи Java SE під загальною публічною ліцензією GNU, версія 2, з Classpath Exception.

Ще один випадок, пов'язаний із заміною процесу результатом, спостерігається при відтворенні *debug*, буквально 'звільняти від помилок' як *налагоджувати*, таким чином, один процес замінено іншим, більш глобальним, що має на меті налагоджену програму: (63) *This allows specialists to implement / test / debug complex, task-specific objects, which you can then trust to run in your own code* (LJ: URL) – Це дозволяє фахівцям впроваджувати / тестувати / налагоджувати складні об'єкти, що стосуються конкретного завдання, які потім можна довіряти виконувати у власному коді.

Заміна однієї властивості іншою, логічно пов’язаною з вихідною, спостерігається при перекладі *pluggability* як *зручність підключення*, хоча *-ability* в англійській мові означає «можливість»: (64) *Pluggability and debugging ease: If a particular object turns out to be problematic, you can simply remove it from your application and plug in a different object as its replacement* (LJ: URL) – Зручність підключення та налагодження: якщо певний об’єкт виявляється проблематичним, ви можете просто видалити його зі своєї програми та підключити інший об’єкт на заміну.

Репрезентація однієї властивості через іншу спостерігається також при відтворенні *verbose mode* як *докладний режим*, де *verbose* ‘багатослівний’ передано як *докладний*: (31) *Start with the verbose mode to get the most feedback while learning the tool* (LJ: URL) – Почніть із докладного режиму, щоб отримати якнайбільше зворотного зв’язку під час вивчення інструмента.

При відтворенні словосполучення *seasoned developer* лексема *seasoned* ‘загартований’ передана більш доцільним для контексту програмування *досвідчений – досвідчений розробник*: (8) *When you become a seasoned Java developer, seeing a class that does not follow this convention will look weird to you* (LJ: URL) – Коли ви станете досвідченим розробником Java, побачити клас, який не відповідає цій умові, здасться вам дивним.

Модуляція також може бути викликаною розбіжностями у репрезентації знань в культурах мови оригіналу та перекладу. Зокрема, в англійській мові десяткові дроби записуються через крапку, в українській – через кому, тому *floating point arithmetic* передається українською як *арифметика з плаваючою комою*: (99) *Floating point arithmetic is a special world in which common operations may behave unexpectedly* (LJ: URL) – Арифметика з плаваючою комою – це особливий світ, у якому звичайні операції можуть вести себе несподівано.

Окрім того, шляхом модуляції деякі описувані в IT дискурсі з програмування дії можуть ставати більш зрозумілими для адресата,

наприклад, *to nail down bugs*, де *nail down* ‘прибивати’ передано як *виправляти*, тобто, переклад відбувається з огляду на особливості слововжитку в мові програмування цільової культури – *виправляти помилки*: (19) *These applications handle the compilation of your source code automatically, they can help you to track errors in the syntax of your Java code and nail down bugs in its execution, among other things* (LJ: URL) – Ці програми, серед іншого, автоматично обробляють компіляцію вашого вихідного коду, вони можуть допомогти вам відслідковувати помилки в синтаксисі вашого коду Java та виправляти помилки під час його виконання.

Таким чином, лексико-семантичні перекладацькі трансформації дозволяють уточнити значення професіоналізмів у контексті програмування ІТ дискурсу при перекладі. Лексико-семантичні трансформації викликані необхідністю зробити текст зрозумілим для цільової аудиторії та адаптувати його до норм слововжитку у мові програмування цільової культури.

3.3 Граматичні перекладацькі трансформації та їх використання при перекладі професіоналізмів у контексті програмування ІТ дискурсу

Лексичні та граматичні параметри мови оригіналу та мови перекладу часто не співпадають, що викликає необхідність застосування при відтворенні англійськомовних професіоналізмів у контексті програмування ІТ дискурсу українською мовою граматичних перекладацьких трансформацій, до яких належать граматичні заміни та додавання.

Під **граматичними замінами** розуміємо заміну слова, що належить до однієї частини мови, словом, що належить до іншої (морфологічні заміни), або ж заміну однієї синтаксичної конструкції іншою (синтаксичні заміни) [42: 113].

Зокрема, морфологічні заміни передбачають заміну частиномовної належності слів при перекладі. Це – такі заміни:

1) заміна іменника прикметником, наприклад, *source code – вихідний код*: (3) *Whereas you can read a source code and understand it, binary or executable files are not meant to be read by a human person* (LJ: URL) – У той час як ви можете прочитати вихідний код і зрозуміти його, двійкові або виконувані файли не призначені для читання людиною; *plain text file – звичайний текстовий файл*: (5) *The first step you need to know is that the Java code you are writing is saved in plain text files* (LJ: URL) – Перший крок, який вам потрібно знати, – це те, що код Java, який ви пишете, зберігається у звичайних текстових файлах; *language shell – мовна оболонка*: (24) *You use the language shell to learn the Java language, explore new features and APIs, and prototype new code* (LJ: URL) – Ви використовуєте мовну оболонку, щоб вивчати мову Java, досліджувати нові функції та API, а також прототипувати новий код; *compile time – час компіляції*: (53) *Preview features are specific to a given Java SE feature release and require the use of special flags at compile time as well as at runtime* (LJ: URL) – Функції попереднього перегляду є специфічними для даного випуску функції Java SE і вимагають використання спеціальних прапорів під час компіляції, а також під час виконання;

2) заміна дієприкметників Participle I та Participle II прикметниками, наприклад, *self-documenting – самодокументований*: (79) *In many cases it will also make your code self-documenting; fields named cadence, speed, and gear, for example, are much more intuitive than abbreviated versions, such as s, c, and g* (LJ: URL) – У багатьох випадках це також зробить ваш код самодокументованим; наприклад, поля з іменами cadence, speed і gear набагато інтуїтивніші, ніж їх скорочені версії, такі як s, c і g; *reserved word – зарезервоване слово*: (80) *Also keep in mind that the name you choose must not be a keyword or reserved word* (LJ: URL) – Також пам’ятайте, що вибране вами ім’я не повинно бути ключовим або зарезервованим словом; *object-oriented programming language – обєктно-орієнтована мова програмування:* (59) *If you’ve never used an object-oriented programming language before, you will need*

to learn a few basic concepts before you can begin writing any code (LJ: URL) – Якщо ви ніколи раніше не використовували об'єктно-орієнтовану мову програмування, вам потрібно буде вивчити кілька основних понять, перш ніж почати писати будь-який код.

Синтаксичні заміни передбачають:

1) заміну складного слова словосполученням, наприклад, *backward-compatibility rules – правила зворотної сумісності:* (51) *Moreover, only final, permanent features are subject to Java's stringent backward-compatibility rules* (LJ: URL) – Більше того, лише остаточні, постійні функції підпадають під суворі правила зворотної сумісності Java; *namespace – простір імен*: (55) *Finally, incubator modules are also shielded from accidental use because incubating can be done only in the jdk.incubator namespace* (LJ: URL) – Нарешті, інкубаційні модулі також захищені від випадкового використання, оскільки інкубація може здійснюватися лише в просторі імен jdk.incubator; *classpath – шлях до класу*: (56) *Therefore, an application on the classpath must use the --add-modules command-line option to explicitly request resolution for an incubating feature* (LJ: URL) – Таким чином, програма на шляху до класу повинна використовувати параметр командного рядка --add-modules, щоб явно запитувати дозвіл для функції інкубації; *code re-use – повторне використання коду*: (62) *Code re-use: If an object already exists (perhaps written by another software developer), you can use that object in your program* (LJ: URL) – Повторне використання коду: якщо об'єкт вже існує (можливо, написаний іншим розробником програмного забезпечення), ви можете використовувати цей об'єкт у своїй програмі; *auto-generated – автоматично згенерований:* (78) *You may find some situations where auto-generated names will contain the dollar sign, but your variable names should always avoid using it* (LJ: URL) – Ви можете зустріти деякі ситуації, коли автоматично згенеровані імена будуть містити знак долара, але в іменах змінних завжди треба уникати його використання;

2) заміну категорії числа мовної одиниці, зокрема, заміна однини множиною, наприклад, *class library – бібліотека класів*: (68) *The Java platform provides an enormous class library (a set of packages) suitable for use in your own applications* (LJ: URL) – Платформа Java надає величезну бібліотеку класів (набір пакетів), придатну для використання у ваших власних програмах.

Трансформація **додавання** полягає у введенні в переклад лексичних елементів, що відсутні в оригіналі, з метою правильної передачі смислу речення (оригіналу), що перекладається, та / або дотримання мовленнєвих і мовних норм, що існують у культурі мови перекладу. Трансформація додавання використовується для компенсації семантичних або граматичних втрат, що мають місце в процесі перекладу [42: 113].

У першу чергу, при перекладі складних слів або іменникових словосполучень часто додаються прийменники, які зв'язують між собою компоненти цих слів чи словосполучень:

– *error message – повідомлення про помилку:* (16) *If it gives you an error message then you need to check your JAVA_HOME and PATH variables as there is most probably something wrong with them* (LJ: URL) – Якщо він видає вам повідомлення про помилку, вам потрібно перевірити свої змінні JAVA_HOME та PATH, оскільки, швидше за все, з ними щось не так;

– *classpath – шлях до класу:* (22) *A class that is part of the core JDK does not need to be added to the classpath to be executed* (LJ: URL) – Клас, який є частиною основного JDK, не потрібно додавати до шляху до класу для виконання;

– *case-sensitive – чутливий до регістру:* (77) *Variable names are case-sensitive* (LJ: URL) – Назви змінних чутливі до регістру;

– *object references – посилання на об'єкт*: (94) *This operator can also be used on objects to assign object references, as discussed in the section Creating Objects* (LJ: URL) – Цей оператор також можна використовувати для об'єктів

для призначення посилань на об’єкт, як ми розповімо в розділі Створення об’єктів.

Додавання також може поставати як засіб пояснювального перекладу, як у наступних випадках:

– *to run – запустити на виконання*: (1) *I understand that you are eager to type some code in your editor and run it to see your first Java application in action!* (LJ: URL) – Я розумію, що ви хочете ввести якийсь код у свій редактор і запустити його на виконання, щоб побачити свою першу програму Java в дії!;

– *ZIP file – файл у форматі ZIP:* (11) *What you get is a ZIP file of about 200MB that you can open with any ZIP utility software* (LJ: URL) – Ви отримуєте файл у форматі ZIP розміром близько 200 МБ, який можна відкрити будь-яким програмним забезпеченням ZIP;

– *statically-typed – мова зі статично типізованими змінними:* (81) *The Java programming language is statically-typed, which means that all variables must first be declared before they can be used* (LJ: URL) – Мова програмування Java – це мова зі статично типізованими змінними, а це означає, що всі змінні повинні бути спочатку оголошені, перш ніж їх можна буде використовувати;

– *reasonable default – розумні значення за замовчуванням:* (83) *Fields that are declared but not initialized will be set to a reasonable default by the compiler* (LJ: URL) – Поля, які оголошені, але не ініціалізовані, будуть встановлені компілятором на розумні значення за замовчуванням.

Окрім того, застосування додавання також може бути викликане відмінностями граматичного ладу мови оригіналу та мови перекладу. Наприклад, *to bookmark* походить від іменника *bookmark* ‘закладка’. В українській мові неможливе використання конверсії в даному контексті, тому перекладач вдається до додавання дієслова, щоб передати лексему мови оригіналу іменником – *додати в закладки*: (72) *Load the page in your browser and bookmark it* (LJ: URL) – Завантажте сторінку у свій браузер і додайте її в закладки.

Отже, основними граматичними засобами відтворення професіоналізмів у контексті програмування IT дискурсу в українському перекладі є граматичні заміни та додавання. Їх застосування зумовлене граматичними відмінностями між мовою оригіналу та мовою перекладу, а також необхідністю пояснення окремих явищ цільовій аудиторії.

Кількісний аналіз способів відтворення при перекладі професіоналізмів IT дискурсу сфери програмування представлено у *Таблиці 3.1*.

Таблиця 3.1

Способи перекладу професіоналізмів у контексті програмування IT дискурсу

Засоби перекладу	**Кількість**	**Частка**
1. Лексичні трансформації	**47**	**42,34%**
нульовий переклад	2	1,8%

Закінчення Таблиці 3.1

Засоби перекладу	**Кількість**	**Частка**
практична транскрипція	7	6,31%
транслітерація	16	14,41%
калькування	22	19,82%
2. Лексико-семантичні трансформації	**29**	**26,13%**
диференціація	12	10,81%
генералізація	5	4,5%
конкретизація	3	2,7%
модуляція	9	8,11%
3. Граматичні трансформації	**35**	**31,53%**
граматичні заміни	24	21,62%
додавання	11	9,91%
Загалом	**111**	**100%**

Таким чином, при відтворенні в українськомовних перекладах професіоналізмів у контексті програмування ІТ дискурсу в українськомовних перекладах найбільш часто використовуються лексичні перекладацькі трансформації (42,34%), здебільшого – калькування (19,82%) та транслітерація (14,41%). Досить часто застосовано граматичні трансформації (31,53%), здебільшого – граматичні заміни (21,62%). Серед лексико-семантичних трансформацій (26,13%) найбільш розповсюдженими є диференціація (10,81%) та модуляція (8,11%).

Висновки до розділу 3

1. Під час відтворення англійськомовних професіоналізмів в контексті програмування ІТ дискурсу українською мовою лексичні перекладацькі трансформації використовуються у 42,34% випадків. Найчастіше такі трансформації представлені калькуванням (19,82%), коли компоненти професіоналізмів мають усталені відповідники українською мовою. Засобами транслітерації (14,41%) та практичної транскрипції (6,31%) відтворюються загальновідомі у світі професіоналізми. Також такі одиниці можуть залишатися у вихідній формі, коли вони є елементами коду, що не перекладається при використанні, або ж власних назв, тоді застосовується нульовий переклад (1,8%).

2. Лексико-семантичні перекладацькі трансформації (26,13%) дозволяють уточнити значення професіоналізмів у контексті програмування ІТ дискурсу при перекладі. Лексико-семантичні трансформації викликані необхідністю зробити текст зрозумілим для цільової аудиторії та адаптувати його до норм слововжитку у мові програмування цільової культури. Серед лексико-семантичних трансформацій найбільш частотними є диференціація (10,81%) та модуляція (8,11%), що передбачають вибір або модифікацію значення слова або словосполучення відповідно до контексту. Генералізація

(4,5%) та конкретизація (2,7%) передбачають відповідно розширення або звуження значення професіоналізмів з метою адаптувати їх до норм слововжитку професійної сфери мови перекладу.

3. Основними граматичними трансформаціями (31,53%), що використовуються при відтворенні професіоналізмів у контексті програмування IT дискурсу в українському перекладі є граматичні заміни (21,62%) та додавання (9,91%). Їх застосування зумовлене граматичними відмінностями між мовою оригіналу та мовою перекладу, а також необхідністю пояснення окремих явищ цільовій аудиторії.

ВИСНОВКИ

Професіоналізми належать до спеціальної лексики, що являє собою слова чи словосполучення, які позначають поняття спеціальної галузі знання або діяльності. Професіоналізми, своєю чергою, – це напівофіційні слова, сферою уживання яких є, як правило, усне побутово-професійне мовлення. Це – слова або звороти, властиві мовленню людей певної професії, здебільшого вони застосовуються в усному неофіційному мовленні людей певного фаху. Професіоналізми виникають у двох випадках: коли певна галузь не має розвиненої термінології, або ж як неофіційні замінники термінів, які вживають у розмовній мові.

Професіоналізми вважаються складною категорією з точки зору перекладу, оскільки сфера їх вживання досить обмежена і вони не зафіксовані в словниках. Переклад професіоналізмів є складним, багатоступеневим процесом, що вимагає глибокого вивчення специфіки галузі використання професійних термінів, контексту та приватних значень слів та виразів.

Професійне мовлення реалізується в користуванні мовою конкретної галузі в усній і писемній формах. У зв’язку з масовою комп’ютеризацією всіх видів діяльності людини увагу мовознавців привертає комунікація у сфері інформаційних технологій; інформатизація суспільства ґрунтується на інформаційних технологіях, їх вплив породжує проблему віртуалізації ціннісних орієнтацій сучасної людини, або віртуалізації реальності, а також формування власне віртуальної реальності. Професійний дискурс програмістів у його письмовій формі – це монолог, актуалізація на письмі мови автора, з погляду прагматики – діалог, проте, не автора із самим собою, а автора з потенційним читачем, точніше уявним співрозмовником, як типовим представником тієї соціально-культурної спільноти, якій автор адресує своє повідомлення. Найбільшу особливість професійного дискурсу інженера-

програміста становить різноманітна в семантичному аспекті спеціальна лексика, що включає слова професіоналізми.

За семантичним критерієм професіоналізми у контексті програмування ІТ дискурсу розподілено на тематичні групи і підгрупи, серед яких найбільш чисельними є підгрупи «процес програмування» (72%), зокрема, «дії в процесі програмування» (16%) та «складові процесу програмування» (13%). 20% професіоналізмів у контексті програмування ІТ дискурсу належать до тематичної групи «програмне забезпечення», де найбільше одиниць містить підгрупа «взаємодія програми з користувачем» (11%). Найменш частотними тематичними групами професіоналізмів у контексті програмування ІТ дискурсу постають «апаратне забезпечення» (4%), «людина» (2%) та «суміжні науки» (2%).

Професіоналізми в контексті програмування ІТ дискурсу представлені практично в рівній мірі словами (42%) та словосполученнями (58%) із кількісною перевагою словосполучень. Словосполучення найчастіше представлені двокомпонентними словосполученнями (50%), переважно – утвореними за схемами N + N (28%) та Adj + N (16%). Серед трикомпонентних словосполучень професіоналізми найчастіше творяться за схемою Adj + N + N (5%). Найбільш продуктивними способами творення професіоналізмів у контексті програмування ІТ дискурсу у формі слів постають афіксація (18%), переосмислення (13%) та словоскладання (7%).

При відтворенні в українськомовних перекладах професіоналізмів у контексті програмування ІТ дискурсу в українськомовних перекладах найбільш часто використовуються лексичні перекладацькі трансформації (42,34%), здебільшого – калькування (19,82%), коли компоненти професіоналізмів мають усталені відповідники українською мовою, та транслітерація (14,41%) при відтворенні загальновідомих у світі професіоналізмів. Практична транскрипція (6,31%) також застосовується при передачі професіоналізмів, що стали інтернаціоналізмами. Окрім того,

професіоналізми можуть залишатися у вихідній формі, коли вони є елементами коду, що не перекладається при використанні, або ж власних назв, тоді застосовується нульовий переклад (1,8%).

Досить часто застосовано граматичні трансформації (31,53%), здебільшого – граматичні заміни (21,62%), застосування яких зумовлене граматичними відмінностями між мовою оригіналу та мовою перекладу. Додавання (9,9%) може бути зумовлене як лексичними і граматичними відмінностями між англійською та українською мовами, так і необхідністю пояснити значення окремої лексичної одиниць більш докладно, щоб зробити текст зрозумілішим для цільової аудиторії.

Лексико-семантичні перекладацькі трансформації (26,13%) дозволяють уточнити значення професіоналізмів у контексті програмування ІТ дискурсу при перекладі. Серед лексико-семантичних трансформацій найбільш частотними є диференціація (10,81%) та модуляція (8,11%), що передбачають вибір або модифікацію значення слова або словосполучення відповідно до контексту. Генералізація (4,5%) та конкретизація (2,7%) передбачають відповідно розширення або звуження значення професіоналізмів з метою адаптувати їх до норм слововжитку професійної сфери мови перекладу.

Загалом, англійськомовні професіоналізми в контексті програмування ІТ дискурсу українською мовою найчастіше відтворюються із застосуванням таких перекладацьких трансформацій, як граматичні заміни (21,62%), калькування (19,82%), транслітерація (14,41%) та диференціація (10,81%).

Перспективами подальших досліджень є типологічне та порівняльне дослідження професіоналізмів у контексті програмування ІТ дискурсу, а також різних галузей техніки англійської та української мов, вивчення стратегій перекладу професіоналізмів у контексті програмування ІТ дискурсу в залежності від цільової аудиторії перекладу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ананко Т. Р. Особливості передачі професійної лексики й корпоративного жаргону у перекладі художнього твору. URL: http://bo0k.net/index.php?p=achapter&bid=19915&chapter=1 (дата звернення: 10.10.2021).

2. Андрощук О. В. Інформаційні технології та їх вплив на розвиток суспільства. За ред. Ю. В. Кондратенко, О. В. Головченко, Т. О. Ворона, М. В. Петрушен. *Збірник наукових праць Центру воєнно-стратегічних досліджень Національного університету оборони України імені Івана Черняховського.* 2014. № 1(50). С. 42–47.

3. Арцишевська А. Л., Кузнецова Л. Р. Фахова юридична мова: особливості викладання на юридичному факультеті. *Молодий вчений.* 2020. № 2.1 (78.1). С. 4–8.

4. Балас І. А., Сидорук Г. І., Професіоналізми як складова професійного мовлення. *Zbiór artykułów naukowych. Konferencji Miedzynarodowej Naukowo-Praktycznej «Filologia, socjologia i kulturoznawstwo. Naukowe Wyszukaj»*. Warszawa: Wydawca: Sp. z o.o. «Diamond trading tour», 2015. С. 33–36.

5. Бережна М. В., Лозовська К. О. Етапи перекладу термінів та професіоналізмів (на матеріалі текстів металургійної тематики). *Science and Education a New Dimension. Philology.* URL: http://seanewdim.com/uploads/3/4/5/1/34511564/httpsdoi.org10.31174send-ph2020-241viii72-01.pdf (дата звернення: 10.10.2021).

6. Берестенко О. Г. Культура професійного комунікація. Луганськ: Вид-во ДЗ «ЛНУ імені Тараса Шевченка», 2013. 299 с.

7. Бугаєвська Ю. В. Професійна лексика як складова фахової мови студентів економічних спеціальностей. *Збірник наукових праць ХНПУ ім. Г. С. Сковороди*. 2017. № 45. С. 179–183.

8. Буйницька О. П. Інформаційні технології та технічні засоби навчання. Київ: Центр учбової літератури, 2012. 240 с.

9. Булик-Верхола С. З., Наконечна Г. В., Теглівець Ю. В. Основи термінознавства. Львів: Видавництво НУ «Львівська політехніка», 2013. 160 с.

10. Велика І. О., Тупахіна О. В. Феномен «Інтернет-дискурс» у сучасній науковій парадигмі. *Science and Education a New Dimension. Philology.* 2017. № 43. С. 61–64.

11. Винник О. Ю., Рубель Н. В. Стратегія залучення читача до співпраці з автором у сучасному англомовному дискурсі програмування. *Наукові записки Національного університету «Острозька академія». Серія: «Філологія».* 2018. № 1 (69), ч. 1. С. 85–88.

12. Вискушенко С. А. Фахова мова як об'єкт лінгвістичного дослідження. *Наукові записки Національного університету «Острозька академія». Серія: Філологічна.* 2015. № 58. С. 142–144.

13. Влахов С., Флорин С. Непереводимое в переводе: учебное пособие. Москва: Международные отношения. 1980. 342 с.

14. Водорєзова С. Р. Поняття та властивості інформації як основи інформаційного суспільства у господарсько-правовому контексті. *Адаптація до права ЄС регулювання економіки України в сучасних умовах: зб. наук. пр. (за матеріалами «Круглого столу», м. Харків, 26 трав. 2015 р.).* Харків, 2015. С. 107–112.

15. Галенко Т. Явище дискурсу в сучасному інформаційному суспільстві. Теорія і методика професійної освіти електронне наукове фахове видання. URL: https://e06d2b5d-7482-48f3-9eee-3163dd30a024.filesusr.com/ugd/2f377b_f50d5c2e7d7e48fcaf77ebe977591c1c.pdf (дата звернення: 22.10.2021)

16. Глущенко А. В. Дискурс як об'єкт філологічної інтерпретації. *Матеріали IV Міжнародної науково-практичної конференції «Сучасні тенденції розвитку освіти і науки в інтердисциплінарному контексті. Діалог*

культур як чинник інтеграції». Електронний збірник матеріалів конференції. Варшава – Ужгород – Херсон, 2019. С. 187–189.

17. Дробязко Ю. І. Особливості англійської науково-технічної термінології. URL: http://www.rusnauka.com/4_SND_2013/Philologia/3_127550.doc.htm (дата звернення: 23.09.2021)

18. Зеленько К. М. Віртуальність як спроба штучного конструювання соціальної дійсності. *Гуманітарний дискурс суспільних проблем: минуле, сучасне, майбутнє: Матеріали Всеукраїнської наукової конференції з міжнародною участю.* Черкаси: ЧІПБ імені Героїв Чорнобиля НУЦЗ України, 2021. С. 63–65.

19. Зубков М. Г. Сучасна українська ділова мова. 3-тє вид., доповнене. Харків: Торсінг, 2003. 448 с.

20. Каламбет Я. І. Лексичні трансформації в науково-технічному перекладі. *Молодий вчений*. 2018. № 4 (2). С. 568–571.

21. Карабан В. І. Переклад англійської наукової і технічної літератури. Вінниця: Нова книга, 2001. 303 с.

22. Карий О. І., Гальків Л. І., Цапулич А. Ю. Розвиток ІТ-сфери України: чинники та напрями активізації. *Journal of Lviv Polytechnic National UniversitySeries of Economics and Management Issues.* 2021.Vol. 5. № 1. С. 42–55.

23. Кацан О. С., Зикун Н. І. Професійний сленг і формування медіатермінології (на прикладі київського бюро радіо свобода). *Тенденції та перспективи формування професійної лексики*. 2020. № 10. С. 182–185.

24. Кияк Т. Вузькогалузеві терміни як основа формування та квазіреферування фахових текстів. *Вісник Національного університету «Львівська політехніка». Серія «Проблеми української термінології».* 2008. № 620. С. 3–5.

25. Кияк Т. Фахові мови як новий напрям лінгвістичного дослідження. *Іноземна філологія*. 2009. № 121. С. 138–141.

26. Кияк Т. Р. Теорія і практика перекладу: підручник для студентів вищих навчальних закладів. За ред. А. М. Науменко, О. Д. Огуй. Вінниця: Нова книга, 2006. 592 с.

27. Кіршо С. М. Інноваційна парадигма забезпечення якості навчання мови за професійним спрямуванням. *Лінгвокультурний дискурс у парадигмі професійної освіти: зб. Матеріалів Міжнар. наук.-практ. конф. Київ, 5 берез. 2015 р.* Київ: КНЕУ, 2015. С. 300–306.

28. Коваленко М. Перекладацький коментар як спосіб відтворення неперекладних елементів задля досягнення адекватності. URL: http://www.masters.kubg.edu.ua/index.php/if/article/viewFile/401/351 (дата звернення: 12.10.2021).

29. Ковтун І. І. Терміни та професіоналізми у правоохоронній сфері. *Лінгвістичні дослідження: зб. наук. пр. ХНПУ*. Харків: ХНПУ ім. Г. Сковороди, 2010. № 29. С. 64–72.

30. Колесник А. О., Бєлікова О. Ф. Перекладацькі прийоми під час перекладу термінології наукових текстів. *Економічна стратегія і перспективи розвитку сфери торгівлі та послуг.* 2010. № 1. С. 719–727.

31. Комарова З. И. Семантическая структура специального слова и ее лексикографическое описание. Свердловск: изд. Уральск. гос. ун-та, 1991. 150 с.

32. Комиссаров В. Н. Теория перевода: Лингвистические аспекты. Москва: Высшая школа, 1990. 253 с.

33. Коновченко О. В. Документування міжнародних договірних відносин. Харків: Національний аерокосмічний університет ім. М. Є. Жуковського «ХАІ», 2012. 63 с.

34. Королева Л. Ю. Профессиональный дискурс программистов в современном обществе. *Вестник ТГТУ*. 2006. Том 12. № 2Б. С. 596–597.

35. Кочан І. Підручники з термінознавства кінця ХХ – початку ХХІ століття. *Термінологія*. 2017. № 869. С. 23–29.

36. Крайтон М. Крылья. Москва: Эксмо-Пресс, 2000. 416 с.

37. Красножан Ж. В. Функції детермінологізованих лексем у художньому тексті. *Науковий вісник Херсонського державного університету*. 2009. № IX. С. 175–177.

38. Крупнов В. Н. Теоретические и практические проблемы перевода. Москва: Р. Валент, 1987. 427 с.

39. Кулічова Н. П. Особливості професійно-сленгової мови журналіста як засобу формування громадянської свідомості. *Доповіді VIII Міжнародного науково-практичного семінару, присвяченого питанням функціонування професійного мовлення, лінгвокультурологічному та соціокультурному аспектам філології.* Ірпінь: УДФС України, 2018. С. 268–270.

40. Лантюхова Н. Н. Термин: Определение понятия и его сущностные признаки. *Вестник Воронежского института ГПС МЧС России.* 2013. № 1 (6) С. 41–44.

41. Макаров М. Л. Основы теории дискурса. Москва: Гнозис, 2003. 280 с.

42. Максімов С. Є. Практичний курс перекладу (англійська та українська мови). Київ: Ленвіт, 2006. 175 с.

43. Михеева С. В. Особенности перевода профессионализмов на примере произведений Артура Хейли. *Современная филология: материалы VI Междунар. науч. конф. Казань, март 2018 г.* Казань: Молодой ученый, 2018. С. 56–60.

44. Міщенко А. Л. Лінгвістика фахових мов та сучасна модель науково-технічного перекладу. Вінниця: Нова Книга, 2013. 448 с.

45. Мостовий М. І. Лексикологія англійської мови. Харків: Основа, 1993. С. 191–192.

46. Онушканич І. В, Цюцьмаць І. В., Штогрин М. В. Особливості та способи перекладу жаргонізмів науково-технічних текстів сфери

комп'ютерних технологій. *Наукові записки Національного університету «Острозька академія». Серія: «Філологічна».* 2015. № 55. С. 273–275.

47. Павлова О. Терміни, професіоналізми і номенклатурні знаки (до проблеми класифікації спеціальної лексики). *Вісник Національного університету «Львівська політехніка». Проблеми української термінології.* 2008. № 620. С. 49–54.

48. Павлюк О. І. Комп'ютерна лексика в сучасній англійській мові. *Сучасний стан та перспективи розвитку науки: матеріали міжнар. студент. наук. конф. Ужгород, 18 груд. 2020 р.* Ужгород: Молодіжна наукова ліга, 2020. Т. 4. С. 85–86.

49. Петрова Н. Е., Чан Динь Ньят Хань Профессиональный дискурс программиста. *Сучасні виклики і актуальні проблеми науки, освіти та виробництва: міжгалузеві диспути (зб. наук. пр.): матеріали VII міжнародної науково-практичної інтернет-конф. Київ, 21 серп. 2020 р.* Київ, 2020. С. 162–164.

50. Петрушова Н. В., Кравченко В. Л., Петрович О. С. Особливості перекладу британських реалій оповідання А. К. Дойла «Скандал у Богемії» українською мовою. *Вісник Харківського національного університету імені В. Н. Каразіна. Серія: «Іноземна філологія. Методика викладання іноземних мов»*. 2021. № 93. С. 43–55.

51. Пивоваров В. М. Мова української юриспруденції. За ред. О. М. Єрахторіна, О. А. Лисенко та ін. Харків: Право, 2020. 330 с.

52. Покровська О. А. Українська термінологія ринкових відносин: дис. на здоб. наук. ступеня канд. філолог. наук: 10.02.01 «Українська мова» / Харківський національний університет ім. В. Н. Каразіна. Харків, 1995. 207 с.

53. Проскуріна Т. В. Українська мова: навчальний посібник для самостійного вивчення дисципліни. Харків: ХІФ УДУФМТ, 2008. 171 с.

54. Прохорова В. Н. Об эмоциональности термина. *Лингвистические проблемы научно-технической терминологии*. Москва: Наука, 1970. С. 153–159.

55. Світлична Є. І., Берестова А. А., Тєлєжкіна О. О. Фахова мова фармацевта. Харків: Тім Пабліш Груп, 2012. 260 с.

56. Селіванова О. О. Сучасна лінгвістика: напрями та проблеми: підручник. Полтава: Довкілля-К, 2008. С. 298.

57. Сложеникина Ю. В. Терминологическая лексика в общеязыковой системе. Самара: Изд-во СамГУ, 2003. 160 с.

58. Слюсаренко О. В. Проблема зв'язку загальновживаної лексики зі спеціальною. *Вісник Запорізького національного університету. Філологічні науки.* 2017. № 2. С. 180–185.

59. Степанов Ю. С. Альтернативный мир, дискурс, факт и принцип причинности. *Язык и наука конца XX века. Сб. статей*. Москва: Институт языкознания РАН, 1995. С. 35–73.

60. Томіленко Л. Термінологічна лексика в сучасній тлумачній лексикографії української літературної мови. Івано-Франківськ: Фоліант, 2015. 160 с.

61. Харицька С. В., Колісниченко А. В. Дослідження функціонування професіоналізмів і лінгвістичних технік інкорпорації фахової мови в текстову структуру словника. *Актуальні питання гуманітарних наук*. 2020. № 30. С. 144–151.

62. Царьова І. В. Мова і термінологія наукових досліджень у юриспруденції: навчальний посібник. Дніпро: Вид. «Інновація», 2019. 114 с .

63. Чередниченко О. І. Теорія і практика перекладу. Київ: Либідь, 2005. 370 с.

64. Черноватий Л. Порівняльні характеристики перекладу складних англомовних галузевих термінів українською мовою. За ред. І. Липко,

С. Романюк. *Наукові записки. Серія: Філологічні науки.* 2020. № 187. С. 557–564.

65. Чернявская В. Е. Дискурс как объект лингвистических исследований. *Текст и дискурс. Проблемы экономического дискурса: сб. науч. тр.* Сант-Петербург, 2001. С. 11–22.

66. Чернявская В. Е. Дискурс власти и власть дискурса: проблемы речевого воздействия: учеб. Пособие. Москва: Флинта: Наука, 2006. 136 с.

67. Шелов С. Д. Терминология, профессиональная лексика и профессионализмы. *Вопросы языкознания*. 1984. № 5. С. 76–87.

68. Шепітько С. В., Тарапатов М. М. Особливості перекладу сучасної медичної термінології. *Вчені записки ТНУ імені В. І. Вернадського. Серія: Філологія. Соціальні комунікації.* 2019. Т. 30 (69). № 1. Ч. 1. С. 175–179.

69. Янчук Т. В. Значення механізму впровадження інформаційних технологій у господарській діяльності підприємств. *Економіка і організація управління*. 2016. № 4 (24). С. 269–276.

70. Auer P. Code-switching in conversation: language, interaction and identity. London, 1993. 355 p.

71. Bargiella-Chiappini F. The language of business: Introduction and Overview. Ed. by S. Harris. *The language of business: an international perspective.* Edinburgh: Edinburgh University Press, 1997. P. 1–18.

72. Crichton M. Airframe. New York: Ballantine Books, 1996. 429 p.

73. Davidson L. Pro SQL Server 2005 Database Design and Optimization. Ed. By L. Davidson, K. Kline, K. Windisch. Apress, 2006. 672 p

74. Hoffmann L. Means of communication. Technical language: An introduction. Tubingen: Gunter Naer, 1985. 307 p.

75. Krpan D. Undergraduate Programming Courses, Students' Perception and Success. *Procedia – Social and Behavioral Sciences*. 2018. Vol. 174. P. 3868–3872.

76. Marguerie F. LINQ in Action. Ed. by F. Marguerie, S. Eichert, J. Wooley. Greenwich: Manning Publications Co., 2008. 572 p.

77. McConnell S. Code Complete: a practical handbook of software construction. Redmond: Microsoft Press, 2004. 960 p.

78. Roelcke T. Jargon. Berlin: Erich Schmidt Verlag GmbH, 2005. 250 p.

СПИСОК ДОВІДКОВОЇ ЛІТЕРАТУРИ

СЛТ – Жеребило Т. В. Словарь лингвистических терминов. Изд. 5-е, испр. и доп. Назрань: ООО «Пилигрим», 2010. 486 с.

CD – Cambridge Dictionary. URL: https://dictionary.cambridge.org/ (дата звернення: 10.01.2022).

SO – What exactly does the letter “Z” refer to in the name ZGC in JDK11? *Stuck Overflow*. URL: https://stackoverflow.com/questions/61853689/what-exactly-does-the-letter-z-refer-to-in-the-name-zgc-in-jdk11 (дата звернення: 10.01.2022).

TP – Technopedia. URL: https://www.techopedia.com/ (дата звернення: 10.01.2022).

СПИСОК ДЖЕРЕЛ ІЛЮСТРАТИВНОГО МАТЕРІАЛУ

LJ – Learn Java. *Java* | *Oracle*. URL: https://dev.java/learn/ (дата звернення: 10.01.2022).

ДОДАТКИ

Додаток А

Приклади професіоналізмів мови програмування та їх відтворення українською мовою

	Речення	Переклад
1.	*I understand that you are eager to type some code in your editor and run it to see your first Java application in action!* (LJ: URL)	Я розумію, що ви хочете ввести якийсь код у свій редактор і запустити його на виконання, щоб побачити свою першу програму Java в дії!
2.	*This transformation is conducted by a special piece of software called a compiler* (LJ: URL).	Це перетворення виконується спеціальним програмним забезпеченням, яке називається «компілятор».
3.	*Whereas you can read a source code and understand it, binary or executable files are not meant to be read by a human person* (LJ: URL).	У той час як ви можете прочитати вихідний код і зрозуміти його, двійкові або виконувані файли не призначені для читання людиною.
4.	*You will see how to download the JDK for free and how to install it later in this tutorial* (LJ: URL).	Ви дізнаєтеся, як безкоштовно завантажити JDK і як його встановити, пізніше в цьому туторіалі.
5.	*The first step you need to know is that the Java code you are writing is saved in plain text files* (LJ: URL).	Перший крок, який вам потрібно знати, – це те, що код Java, який ви пишете, зберігається у звичайних текстових файлах.
6.	*If this technical term does not mean anything to you, do not worry, all you*	Якщо цей технічний термін для вас нічого не означає, не

	need to remember at this point is that all the code you write must be held in a Java class (LJ: URL).	хвилюйтеся, все, що вам потрібно пам'ятати, – це те, що весь код, який ви пишете, повинен зберігатися в класі Java.
7.	*A Java class is created by a special declaration in a text file* (LJ: URL).	Клас Java створюється за допомогою спеціального «оголошення» в текстовому файлі.
8.	*When you become a seasoned Java developer, seeing a class that does not follow this convention will look weird to you* (LJ: URL).	Коли ви станете досвідченим розробником Java, побачити клас, який не відповідає цій умові, здасться вам дивним.
9.	*It is a set of tools and libraries to create enterprise-class applications* (LJ: URL).	Це набір інструментів і бібліотек для створення додатків корпоративного класу.
10.	*This page provides production-ready open-source builds of the Java Development Kit, an implementation of the Java SE Platform under the GNU General Public License, version 2, with the Classpath Exception* (LJ: URL).	На цій сторінці представлені готові до використання збірки Java Development Kit, реалізація платформи Java SE під загальною публічною ліцензією GNU, версія 2, з Classpath Exception.
11.	*What you get is a ZIP file of about 200MB that you can open with any ZIP utility software* (LJ: URL).	Ви отримуєте файл у форматі ZIP розміром близько 200 МБ, який можна відкрити будь-яким програмним забезпеченням ZIP.
12.	*You can unzip the content of this file anywhere on your computer* (LJ: URL).	Ви можете розпакувати вміст цього файлу будь-де на своєму комп'ютері.

13.	*Once this is done you need to create an environment variable called JAVA_HOME that points to the directory where you unzipped the JDK* (LJ: URL).	Після цього вам потрібно створити змінну середовища під назвою JAVA_HOME, яка вказує на каталог, де ви розпакували JDK.
14.	*First you need to open a DOS prompt* (LJ: URL).	Спочатку вам треба відкрити командний рядок.
15.	*The exact directory depends on the distribution file you have expanded* (LJ: URL).	Точний каталог залежить від файлу дистрибутиву, який ви розгорнули.
16.	*If it gives you an error message then you need to check your JAVA_HOME and PATH variables as there is most probably something wrong with them* (LJ: URL).	Якщо він видає вам повідомлення про помилку, вам потрібно перевірити свої змінні JAVA_HOME та PATH, оскільки, швидше за все, з ними щось не так.
17.	*So far your class is empty; there is no executable code in it* (LJ: URL).	Поки що ваш клас порожній; в ньому немає виконуваного коду.
18.	*Developers that work on large applications do not use plain text editors to manage their source code; they use Integrated Development Environments* (LJ: URL).	Розробники, які працюють над великими програмами, не використовують звичайні текстові редактори для керування вихідним кодом; вони використовують інтегровані середовища розробки.
19.	*These applications handle the compilation of your source code automatically, they can help you to track errors in the syntax of your Java code and nail down bugs in its*	Ці програми, серед іншого, автоматично обробляють компіляцію вашого вихідного коду, вони можуть допомогти вам відслідковувати помилки в синтаксисі вашого коду Java та

	execution, among other things (LJ: URL).	виправляти помилки під час його виконання.
20.	*This works by the java launcher automatically invoking the compiler and storing the compiled code in-memory* (LJ: URL).	Це працює завдяки тому, що програма запуску Java автоматично викликає компілятор і зберігає скомпільований код у пам'яті.
21.	*Multiple classes can be defined within the same source file if needed for encapsulation purposes, like in this example* (LJ: URL).	Декілька класів можна визначити в одному вихідному файлі, якщо це необхідно для цілей інкапсуляції, як у цьому прикладі.
22.	*A class that is part of the core JDK does not need to be added to the classpath to be executed* (LJ: URL).	Клас, який є частиною основного JDK, не потрібно додавати до шляху до класу для виконання.
23.	*On a Unix-like operating system, a single-file source-code application, can also be launched as a shebang file like a script* (LJ: URL).	У Unix-подібній операційній системі однофайлова програма з вихідним кодом також може бути запущена файлом шебанг як сценарій.
24.	*You use the language shell to learn the Java language, explore new features and APIs, and prototype new code* (LJ: URL).	Ви використовуєте мовну оболонку, щоб вивчати мову Java, досліджувати нові функції та API, а також прототипувати новий код.
25.	*Command-line scripts are run after startup scripts* (LJ: URL).	Скрипти командного рядка запускаються після сценаріїв запуску.
26.	*To run a script after jshell is started, use the /open command* (LJ: URL).	Щоб запустити скрипт після запуску jshell, скористайтеся командою /open.

27.	*To accept input from standard input and suppress the interactive I/O, enter a hyphen (-) for load-files* (LJ: URL).	Щоб прийняти стандартне введення та придушити інтерактивний ввід-вивід, введіть дефіс (-) для завантажуваних файлів.
28.	*This option enables the use of the jshell tool in pipe chains* (LJ: URL).	Цей параметр дозволяє використовувати інструмент jshell у ланцюгах каналів.
29.	*Java statements, variable definitions, method definitions, class definitions, import statements, and expressions are accepted* (LJ: URL).	Приймаються оператори Java, визначення змінних, визначення методів, визначення класів, оператори імпорту та вирази.
30.	*The bits of code entered are called snippets* (LJ: URL).	Введені біти коду називаються сніппетами.
31.	*Start with the verbose mode to get the most feedback while learning the tool* (LJ: URL).	Почніть із докладного режиму, щоб отримати якнайбільше зворотного зв’язку під час вивчення інструмента.
32.	*Use this option to run only the scripts entered on the command line when JShell is started, or to start JShell without any preloaded information if no scripts are entered* (LJ: URL).	Використовуйте цей параметр, щоб запускати лише сценарії, введені в командному рядку під час запуску JShell, або запускати JShell без попередньо завантаженої інформації, якщо сценарії не введені.
33.	*Environment settings entered on the command line or provided with a previous /reset, /env, or /reload command are maintained unless an*	Параметри середовища, введені в командному рядку або надані попередньою командою /reset, /env або /reload, зберігаються, якщо не

	option is entered that overwrites the setting (LJ: URL).	введено параметр, який перезаписує налаштування.
34.	*This option overrides the path in the CLASSPATH environment variable* (LJ: URL).	Цей параметр замінює шлях у змінній середовища CLASSPATH.
35.	*Sets the editor to the default editor provided with JShell. This option can't be used if a command for starting an editor is entered* (LJ: URL).	Встановлює для редактора стандартні налаштування, надані JShell. Цей параметр не можна використовувати, якщо введена команда для запуску редактора.
36.	*When entering snippets, commands, subcommands, command arguments, or command options, use the Tab key to automatically complete the item* (LJ: URL).	Під час введення сніппетів, команд, підкоманд, аргументів команди або параметрів команди використовуйте клавішу Tab, щоб автоматично завершити елемент.
37.	*When entering a method call, use the Tab key after the method call's opening parenthesis to see the parameters for the method* (LJ: URL).	Під час введення виклику методу використовуйте клавішу Tab після відкриваючої дужки виклику методу, щоб побачити параметри методу.
38.	*The ID for each snippet begins with the letter s, which indicates it's a startup snippet* (LJ: URL).	Ідентифікатор кожного фрагмента починається з літери s, яка вказує, що це фрагмент сценарію запуску.
39.	*Note that a scratch variable is automatically created to hold the result because no variable was provided* (LJ: URL).	Зауважте, що змінна скретчу створюється автоматично для збереження результату, оскільки змінна не була надана.
40.	*With millions of people relying on Java in production for critical*	Оскільки мільйони людей покладаються на Java у створенні

	workloads, the reach of Java is global (LJ: URL).	програмного забезпечення, що могло б витримувати критичні робочі навантаження, використання Java є глобальним.
41.	*It is therefore critical to provide developers with preliminary access to new features to encourage feedback – feedback that will help refine those features and reach the expected quality level for their final and permanent form* (LJ: URL).	Тому дуже важливо надати розробникам попередній доступ до нових функцій, щоб заохочувати зворотний зв'язок – зворотний зв'язок, який допоможе вдосконалити ці функції та досягти очікуваного рівня якості для їх остаточної та постійної форми.
42.	*Incubating (also known as incubator modules), for potentially new APIs and JDK tools* (LJ: URL).	Інкубаційні (також відомі як інкубаційні модулі) для потенційно нових API та інструментів JDK.
43.	*In addition, there are other nonfinal features that don't fit in any of those three categories, that you will see later* (LJ: URL).	Крім того, є інші нефінальні функції, що не вписуються в жодну з цих трьох категорій, і їх ви побачите пізніше.
44.	*Switch expressions is a language feature* (LJ: URL).	«Switch»-вирази – це функція мови.
45.	*The Z garbage collector is a HotSpot feature* (LJ: URL).	Збірник сміття Z – це функція HotSpot.
46.	*The HTTP/2 Client API is an API added as an incubating feature in Java SE 9 and Java SE 10, and was*	Клієнтський API HTTP/2 – це API, доданий як інкубаційна функція в Java SE 9 і Java SE 10, а потім він

	then made a standard feature in Java SE 11 (LJ: URL).	став стандартною функцією в Java SE 11.
47.	*A Java SE API feature in the core Java platform, such as java.lang.Object, java.lang.String, or java.io.File* (LJ: URL).	Функція API Java SE в основній платформі Java, наприклад java.lang.Object, java.lang.String або java.io.File.
48.	*If there is a high demand for an enhancement, if it impacts the JDK or the processes and infrastructure by which the JDK itself is developed (such as moving the JDK source code repository to GitHub), or simply because it requires quite a bit of engineering investment, then it is nontrivial* (LJ: URL).	Якщо є високий попит на покращення, якщо це впливає на JDK або процеси та інфраструктуру, за допомогою яких розробляється сам JDK (наприклад, переміщення сховища вихідного коду JDK на GitHub), або просто тому, що це вимагає значних інвестицій в інженерію, то це нетривіально.
49.	*There might be one or multiple iterations of the preliminary access phase during which you have access to nonfinal features* (LJ: URL).	Може бути одна або кілька ітерацій фази попереднього доступу, під час якої ви маєте доступ до нефінальних функцій.
50.	*An experimental feature is a test-bed mechanism used to gather feedback on nontrivial HotSpot enhancements* (LJ: URL).	Експериментальною функцією є механізм тестового стенда, який використовується для збору відгуків про нетривіальні покращення HotSpot.
51.	*Moreover, only final, permanent features are subject to Java's stringent backward-compatibility rules* (LJ: URL).	Більше того, лише остаточні, постійні функції підпадають під суворі правила зворотної сумісності Java.

52.	*Therefore, to avoid unintentional use, preview and experimental features are disabled by default, and the JDK documentation unequivocally warns you about the nonfinal nature of these features and any of their associated APIs* (LJ: URL).	Тому, щоб уникнути ненавмисного використання, попередній перегляд і експериментальні функції вимкнені за замовчуванням, а документація JDK однозначно попереджає вас про неостаточний характер цих функцій і будь-яких пов’язаних з ними API.
53.	*Preview features are specific to a given Java SE feature release and require the use of special flags at compile time as well as at runtime* (LJ: URL).	Функції попереднього перегляду є специфічними для даного випуску функції Java SE і вимагають використання спеціальних прапорів під час компіляції, а також під час виконання.
54.	*You need a first mandatory Square class, which you need to store in the same package or module as the Shape interface* (LJ: URL).	Вам потрібен перший обов’язковий клас Square, який потрібно зберігати в тому ж пакеті або модулі, що й інтерфейс Shape.
55.	*Finally, incubator modules are also shielded from accidental use because incubating can be done only in the jdk.incubator namespace* (LJ: URL).	Нарешті, інкубаційні модулі також захищені від випадкового використання, оскільки інкубація може здійснюватися лише в просторі імен jdk.incubator.
56.	*Therefore, an application on the classpath must use the --add-modules command-line option to explicitly request resolution for an incubating feature* (LJ: URL).	Таким чином, програма на шляху до класу повинна використовувати параметр командного рядка --add-modules, щоб явно запитувати дозвіл для функції інкубації.

57.	*The goal of those projects is to conduct fundamental investigations in particular areas to drastically improve (or completely revamp) certain aspects of the Java platform* (LJ: URL).	Метою цих проектів є проведення фундаментальних досліджень у певних областях, щоб радикально покращити (або повністю оновити) певні аспекти платформи Java.
58.	*The unique goal of such occasional feature-specific early access (EA) JDK builds is solely to allow expert users to test specific novel features early* (LJ: URL).	Унікальна мета таких випадкових збірок JDK для специфічного раннього доступу (early access – EA) полягає виключно в тому, щоб дозволити експертним користувачам завчасно тестувати конкретні нові функції.
59.	*If you've never used an object-oriented programming language before, you will need to learn a few basic concepts before you can begin writing any code* (LJ: URL).	Якщо ви ніколи раніше не використовували об'єктно-орієнтовану мову програмування, вам потрібно буде вивчити кілька основних понять, перш ніж почати писати будь-який код.
60.	*This section explains how state and behavior are represented within an object, introduces the concept of data encapsulation, and explains the benefits of designing your software in this manner* (LJ: URL).	У цьому розділі пояснюється, як стан і поведінка представлені в об'єкті, представлена концепція інкапсуляції даних та пояснюються переваги проектування програмного забезпечення саме таким чином.
61.	*Hiding internal state and requiring all interaction to be performed through an object's methods is*	Приховування внутрішнього стану та вимога, щоб усі взаємодії виконувались за допомогою

	known as data encapsulation – a fundamental principle of object-oriented programming (LJ: URL).	методів об’єкта, відомі як інкапсуляція даних – фундаментальний принцип об’єктно-орієнтованого програмування.
62.	*Code re-use: If an object already exists (perhaps written by another software developer), you can use that object in your program* (LJ: URL).	Повторне використання коду: якщо об’єкт вже існує (можливо, написаний іншим розробником програмного забезпечення), ви можете використовувати цей об’єкт у своїй програмі.
63.	*This allows specialists to implement / test / debug complex, task-specific objects, which you can then trust to run in your own code* (LJ: URL).	Це дозволяє фахівцям впроваджувати / тестувати / налагоджувати складні об’єкти, що стосуються конкретного завдання, які потім можна довіряти виконувати у власному коді.
64.	*Pluggability and debugging ease: If a particular object turns out to be problematic, you can simply remove it from your application and plug in a different object as its replacement* (LJ: URL).	Зручність підключення та налагодження: якщо певний об’єкт виявляється проблематичним, ви можете просто видалити його зі своєї програми та підключити інший об’єкт на заміну.
65.	*In the Java programming language, each class is allowed to have one direct superclass, and each superclass has the potential for an*	У мові програмування Java кожному класу дозволено мати один прямий надклас, і кожен

	unlimited number of subclasses (LJ: URL).	надклас може мати необмежену кількість підкласів.
66.	*At the beginning of your class declaration, use the extends keyword, followed by the name of the class to inherit from* (LJ: URL).	На початку оголошення класу використовуйте ключове слово extends, а потім ім’я класу, від якого слід успадкувати.
67.	*Interfaces form a contract between the class and the outside world, and this contract is enforced at build time by the compiler* (LJ: URL).	Інтерфейси утворюють узгодження між класом і зовнішнім світом, і це узгодження виконується компілятором під час збірки.
68.	*The Java platform provides an enormous class library (a set of packages) suitable for use in your own applications* (LJ: URL).	Платформа Java надає величезну бібліотеку класів (набір пакетів), придатну для використання у ваших власних програмах.
69.	*Its packages represent the tasks most commonly associated with general-purpose programming* (LJ: URL).	Його пакети представляють завдання, які найчастіше пов'язані з програмуванням загального призначення.
70.	*A Socket object allows for the creation and use of network sockets* (LJ: URL).	Об’єкт Socket дозволяє створювати та використовувати мережеві сокети.
71.	*Various GUI objects control buttons and check boxes and anything else related to graphical user interfaces* (LJ: URL).	Різні об’єкти GUI управляють кнопками, прапорцями та всим іншим, що стосується графічних інтерфейсів користувача.
72.	*Load the page in your browser and bookmark it* (LJ: URL).	Завантажте сторінку у свій браузер і додайте її в закладки.

73.	*Do fields have to be initialized when they are declared?* (LJ: URL)	Чи потрібно ініціалізувати поля під час їх оголошення?
74.	*Non-static fields are also known as instance variables because their values are unique to each instance of a class (to each object, in other words)* (LJ: URL).	Нестатичні поля також відомі як змінні екземпляра, оскільки їх значення є унікальними для кожного екземпляра класу (іншими словами, для кожного об'єкта).
75.	*Similar to how an object stores its state in fields, a method will often store its temporary state in local variables* (LJ: URL).	Подібно до того, як об'єкт зберігає свій стан у полях, метод часто зберігає свій тимчасовий стан у локальних змінних.
76.	*This applies to other parameter-accepting constructs as well (such as constructors and exception handlers) that you'll learn about later in the tutorial* (LJ: URL).	Це також стосується інших конструкцій, що приймають параметри (таких як конструктори та обробники винятків), про які ви дізнаєтеся пізніше.
77.	*Variable names are case-sensitive* (LJ: URL).	Назви змінних чутливі до регістру.
78.	*You may find some situations where auto-generated names will contain the dollar sign, but your variable names should always avoid using it* (LJ: URL).	Ви можете зустріти деякі ситуації, коли автоматично згенеровані імена будуть містити знак долара, але в іменах змінних завжди треба уникати його використання.
79.	*In many cases it will also make your code self-documenting; fields named cadence, speed, and gear, for example, are much more intuitive*	У багатьох випадках це також зробить ваш код самодокументованим; наприклад, поля з іменами cadence, speed і gear

	than abbreviated versions, such as s, c, and g (LJ: URL).	набагато інтуїтивніші, ніж їх скорочені версії, такі як s, c і g.
80.	*Also keep in mind that the name you choose must not be a keyword or reserved word* (LJ: URL).	Також пам'ятайте, що вибране вами ім'я не повинно бути ключовим або зарезервованим словом.
81.	*The Java programming language is statically-typed, which means that all variables must first be declared before they can be used* (LJ: URL).	Мова програмування Java – це мова зі статично типізованими змінними, а це означає, що всі змінні повинні бути спочатку оголошені, перш ніж їх можна буде використовувати.
82.	*String objects are immutable, which means that once created, their values cannot be changed* (LJ: URL).	Рядкові об'єкти незмінні, що означає, що після створення їх значення не можна змінити.
83.	*Fields that are declared but not initialized will be set to a reasonable default by the compiler* (LJ: URL).	Поля, які оголошені, але не ініціалізовані, будуть встановлені компілятором на розумні значення за замовчуванням.
84.	*Relying on such default values, however, is generally considered bad programming style* (LJ: URL).	Однак використання таких значень за замовчуванням зазвичай вважається поганим стилем програмування.
85.	*Local variables are slightly different; the compiler never assigns a default value to an uninitialized local variable* (LJ: URL).	Локальні змінні дещо відрізняються; компілятор ніколи не призначає значення за замовчуванням неініціалізованій локальній змінній.

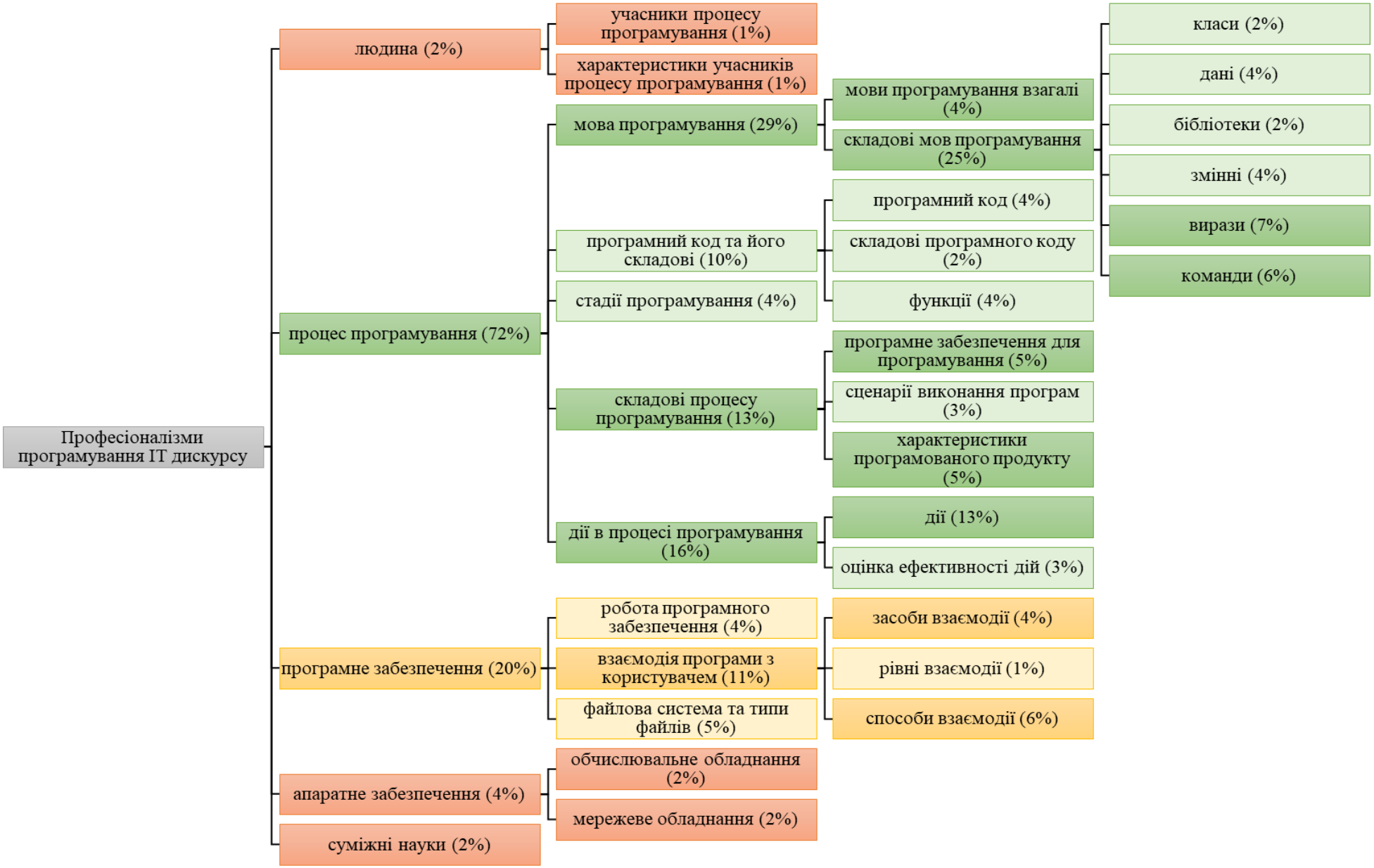
86.	*Accessing an uninitialized local variable will result in a compile-time error* (LJ: URL).	Спроба звернутися до неініціалізованої локальної змінної призведе до помилки під час компіляції.
87.	*Primitive types are special data types built into the language; they are not objects created from a class* (LJ: URL).	Примітивні типи – це спеціальні типи даних, вбудовані в мову; вони не є об'єктами, створеними з класу.
88.	*A literal is the source code representation of a fixed value; literals are represented directly in your code without requiring computation* (LJ: URL).	Літералом є вихідне представлення фіксованого значення; літерали представлені безпосередньо у вашому коді, не вимагаючи обчислень.
89.	*You can only use it for local variables declared in methods, constructors and initializer blocks* (LJ: URL).	Ви можете використовувати його лише для локальних змінних, оголошених у методах, конструкторах та блоках ініціалізації.
90.	*Since null has no type, the variable must have an initializer* (LJ: URL).	Оскільки null не має типу, змінна повинна мати ініціалізатор.
91.	*Learning the operators of the Java programming language is a good place to start* (LJ: URL).	Вивчення операторів мови програмування Java – гарне місце для початку.
92.	*When operators of equal precedence appear in the same expression, a rule must govern which is evaluated first* (LJ: URL).	Коли в одному виразі з'являються оператори однакового пріоритету, повинно бути правило, що визначає, який оператор використовується першим.

93.	*In general-purpose programming, certain operators tend to appear more frequently than others; for example, the assignment operator = is far more common than the unsigned right shift operator >>>* (LJ: URL).	У програмуванні загального призначення деякі оператори мають тенденцію з'являтися частіше, ніж інші; наприклад, оператор присвоєння = набагато частіше зустрічається, ніж беззнаковий оператор зсуву вправо >>>.
94.	*This operator can also be used on objects to assign object references, as discussed in the section Creating Objects* (LJ: URL).	Цей оператор також можна використовувати для об’єктів для призначення посилань на об’єкт, як ми розповімо в розділі Створення об’єктів.
95.	*The unary operators require only one operand; they perform various operations such as incrementing / decrementing a value by one, negating an expression, or inverting the value of a boolean* (LJ: URL).	Унарні оператори вимагають лише однієї операнди; вони виконують різні операції, такі як збільшення / зменшення значення на одиницю, заперечення виразу або інвертування логічного значення.
96.	*This operator is also known as the ternary operator because it uses three operands* (LJ: URL).	Цей оператор також відомий як трійковий оператор, оскільки він використовує три операнди.
97.	*The Java programming language also provides operators that perform bitwise and bit shift operations on integral types* (LJ: URL).	Мова програмування Java також надає оператори, які виконують операції порозрядного та бітового зсуву над інтегральними типами.
98.	*An expression is a construct made up of variables, operators, and method*	Вираз – це конструкція, що складається зі змінних, операторів

	invocations, which are constructed according to the syntax of the language, that evaluates to a single value (LJ: URL).	і викликів методів, які створюються відповідно до синтаксису мови, що обчислюється до одного значення.
99.	*Floating point arithmetic is a special world in which common operations may behave unexpectedly* (LJ: URL).	Арифметика з плаваючою комою – це особливий світ, у якому звичайні операції можуть вести себе несподівано.
100.	*A statement forms a complete unit of execution* (LJ: URL).	Оператор утворює повну одиницю виконання.

Додаток Б

Тематичні групи професіоналізмів у контексті програмування IT дискурсу



SUMMARY

The master's qualification paper deals with the study of the peculiarities of the rendering in translation professionalism in the context of IT discourse on programming based on the example of Java programming materials.

The relevance of the study is explained by the insufficient study of the problem, as well as by the interest of linguists and translators in the problems of linguistic units of the discourse of professional communication and the need to determine the ways of translating professional vocabulary in the context of IT discourse on programming. In addition, the relevance of the research is also determined by the increasingly rapid growth of international communication in the field of programming, which requires solving practical problems in order to improve the quality of such communication.

The aim of the study is to analyze the strategies and specific means of rendering in translation into Ukrainian English professionalisms in the context of IT discourse on programming.

Achieving the aim of the research involves fulfilling the following **objectives**:

1) to consider professionalism as an object of linguistic research;

2) to investigate professionalism as a problem of translation studies;

3) to determine the peculiarities of representing programming activities in the discourse of information technologies;

4) to analyze the semantic parameters of professionalism in the context of IT discourse on programming;

5) to identify the peculiarities of forming professionalism in the context of IT discourse on programming;

6) to highlight lexical translation transformations used when translating professionalisms in the context of IT discourse on programming;

7) to analyze lexical and semantic translation transformations as a means of transferring in translation professionalisms in the context of IT discourse on programming;

8) to characterize grammatical translation transformations and their use in the course of translating professionalisms in the context of IT discourse on programming.

The object of the research is professionalism in the context of IT discourse on programming in English texts and their translations into Ukrainian.

The subject of the research is structural and semantic features, and translation transformations used in the process of rendering in Ukrainian translation English professionalism in the context of IT discourse on programming.

The material of the research is 100 text fragments extracted by continuous sampling from the text devoted to the Java programming language on the official website of its developer Sun Microsystems as a division of the Oracle company. The total amount of analyzed professionalisms is 100 units; in the course of the study, the use of 111 translation transformations was analyzed.

The main **methods** involved in the process of the research were the continuous sampling method, which made it possible to select the illustrative research material; methods of semantic and component analysis allowed to carry out a comprehensive analysis of professional vocabulary from the point of view of linguistics; transformational analysis was used to identify the means of reproducing professionalism in translation; summarization of information was carried out using methods of quantitative analysis.

The scientific novelty of the conducted research lies in the fact that the thematic classification of professionalism in the context of IT discourse on programming was developed and its characteristic features were determined. In addition, the work proposes structural analysis of professionalism in the context of IT discourse on programming, namely, the structure of these linguistic units was determined according to the number of components, and they were further divided into groups according to the means of forming. In addition, the research defines the main transformations used for rendering in Ukrainian translation English professionalism in the context of IT discourse on programming.

The practical significance of the research is determined by the fact that the study of the structural and semantic features of professionalism in the context of IT discourse on programming and the features of their rendering in translation is a contribution to the

theory of comparative linguistics, as well as to the theory of interlanguage contacts, languages for special purposes, and discourse science.

The results of the conducted research can be implemented in the teaching English lexicology courses and Aspect Translation. The obtained information can also be used in lexicography with the aim of codifying and unifying professional vocabulary in the field of programming and in creating explanatory or bilingual dictionaries of professionalisms.

The scope and structure of the research paper. The qualification paper consists of an Introduction, three Chapters, Conclusions, Bibliography, List of Reference Sources, List of Data Sources, two Annexes and Summary in English.

Research results. Professionalisms belong to special vocabulary, which is words or phrases that denote the concept of a special field of knowledge or activity. Professionalisms, in turn, are semi-official words, the sphere of use of which is, as a rule, oral everyday and professional communication. These are words or phrases characteristic of the speech of people belonging to certain profession; mostly they are used in oral informal speech of people of a certain profession. Professionalisms arise in two cases: when a certain field does not have a developed terminology, or as unofficial substitutes for terms that are used in colloquial language.

Professionalisms are considered a difficult category from the point of view of translation, since the scope of their use is quite limited, and they are not recorded in dictionaries. Translation of professional units is a complex, multi-stage process that requires a deep study of the specifics of the field of use of professional units, context and private meanings of words and expressions.

Professional speech is realized in the use of the language of a specific field in oral and written forms. The professional discourse of programmers in its written form is a monologue, the actualization of the author’s language in writing; from the point of view of pragmatics, it is a dialogue, however, not by the author with oneself, but by the author with a potential reader, or rather an imaginary interlocutor, as a typical representative of that sociocultural community, to whom the author addresses one’s message. The most

characteristic feature of the professional discourse of a software engineer is the semantically diverse special vocabulary, which includes professionalisms.

According to the semantic criterion, professionalisms in the context of IT discourse on programming are divided into thematic groups and subgroups, among which the most numerous groups are the "programming process" (72%), in particular, "actions in programming process" (16%) and "components of programming process" (13%). 20% of professionalisms in the context of IT discourse on programming belong to the thematic group "software", where the subgroup "interaction of program with user" contains the most of units (11%). The least frequent thematic groups of professionalism in the context of IT discourse on programming are "hardware" (4%), "human-being" (2%) and "related sciences" (2%).

Professionalisms in the context of IT discourse on programming are almost equally represented by words (42%) and word combinations (58%), with a quantitative advantage of word combinations. Word combinations are most often represented by two-component ones (50%), mainly – those formed according to the schemes N + N (28%) and Adj + N (16%). Among three-component word combinations, professionalisms are most often created according to the scheme Adj + N + N (5%). The most productive ways of creating professionalism in the context of IT discourse on programming in the form of words are affixation (18%), rethinking of meaning (13%) and compounding (7%).

When reproducing in Ukrainian translations professionalisms in the context of IT discourse on programming, lexical translation transformations (42.34%) are most often used, mostly loan translation (19.82%) when components of professionalisms have established counterparts in Ukrainian, and transliteration (14 .41%) when reproducing well-known international professionalisms. Practical transcription (6.31%) is also used in the rendering professionalisms that have become internationalisms. In addition, professionalisms can remain in their original form when they are elements of code that is not translated when used, or of proper names; then zero translation is applied (1.8%).

Grammatical transformations are used quite often (31.53%), mostly – grammatical replacements (21.62%), the use of which is caused by the grammatical differences

between the source language and the target language. Addition (9.9%) is used because of both lexical and grammatical differences between English and Ukrainian, as well as due to the need to explain the meaning of individual lexical units in more detail to make the text more understandable for the target audience.

Lexical and semantic translation transformations (26.13%) allow clarifying the meaning of professionalism in the context of IT discourse on programming in translation. Among the lexical and semantic transformations, the most frequent ones are differentiation (10.81%) and modulation (8.11%) which involve choosing or modifying the meaning of a word or word combination according to the context. Generalization (4.5%) and substantiation (2.7%) provide, respectively, widening or narrowing of the meaning of professionalisms in order to adapt them to the norms of word usage in the professional sphere of the target language.

In general, English professionalisms in the context of IT discourse on programming are most often reproduced in the Ukrainian language using such translation transformations as grammatical replacements (21.62%), loan translation (19.82%), transliteration (14.41%) and differentiation (10.81 %).

The prospects for further research include typological and comparative study of professionalism in the context of IT discourse on programming, as well as in various technical branches of the English and Ukrainian languages, the study of translation strategies of rendering professionalism in the context of IT discourse on programming depending on the target audience of such translation.