БАКАЛАВРА РОБОТА

БР. ІП - 24.00.00.000 ПЗ

Група ІП-21-4

Цариняк Любомир

2025

Івано-Франківський національний технічний університет нафти і газу
Інститут інформаційних технологій
Кафедра інженерії програмного забезпечення

Цариняк Любомир Тарасович

(прізвище, ім'я, по батькові)

УДК 004.942
(індекс)

БАКАЛАВРСЬКА РОБОТА

Стратегії модернізації та інтеграції застарілого коду

(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121– Інженерія програмного забезпечення

(шифр і назва спеціальності)

Робота містить результати власних досліджень, використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело:

Здобувач освітнього ступеня _____Цариняк Л. Т._____
(підпис, ініціали та прізвище здобувача)

Науковий керівник _____Процюк Василь Романович, к.т.н., доцент ____
(підпис, прізвище, ім'я, по батькові, науковий ступінь, вчене звання керівника)

Допущено до захисту
Завідувач кафедри

_доц._____Бандура В.В.___
(посада) (підпис) (дата) (ініціали та прізвище)

Івано-Франківськ – 2025

Івано-Франківський національний технічний університет нафти і газу

Інститут, факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Ступінь вищої освіти бакалавр

Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедрою **ІПЗ**
доцент. В.В. Бандура
“ ____ ” _____2025 р.

ЗАВДАННЯ
НА ДИПЛОМНУ РОБОТУ БАКАЛАВРА СТУДЕНТОВІ

Цариняк Любомир Тарасович

(прізвище, ім’я, по-батькові)

1. Тема проекту (роботи) "Стратегії модернізації та інтеграції застарілого коду”.

керівник проекту (роботи) к.т.н., доц. Процюк В.Р.

затверджені наказом вищого навчального закладу від “ 28 ” квітня 2025 р. № 264/7

2. Строк подання студентом проекту (роботи) 10 червня 2025 р.

3. Вихідні дані до проекту (роботи) Результати і матеріали отримані під час проходження переддипломної практики

4. Зміст розрахунково - пояснювальної записки(перелік питань, які потрібно розробити)

1 Вступ до проблеми

2 Огляд існуючих концепцій , рішень та сервісів в даній області

3 Побудова моделі або алгоритму власного рішення

4 Документування реалізації власного оригінального рішення вибраними засобами

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

1. Порівняння стратегій модернізації щодо вартості та повторного використання (рис. 1.1, ст 16)

2. Типи діяльності та концепції (рис. 2.1, ст 43)

3. Граф залежностей (рис. 2,5, ст 50)

4 Структура оцінювання (рис. 3,2, ст 72)

5. Розподіл методів і технік, що використовуються для визначення доцільності еволюції (рис. 3,7, ст 82)

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

2. Дата видачі завдання 10 травня 2025 р.

Керівник _____
(підпис)

Завдання прийняв до виконання_____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту	Примітка
1	Визначення та обгрунтування теми роботи	15.02.2025	виконано
2	Огляд існуючих концепцій, рішень та сервісів в даній області	25.02.2025	виконано
3	Побудова моделі або алгоритму власного рішення	15.03.2025	виконано
4	Документування реалізації власного оригінального рішення вибраними засобами	25.04.2025	виконано
5	Оформлення пояснювальної записки кваліфікаційної роботи	10.06.2025	виконано

Студент _____
(підпис)

Керівник роботи _____
(підпис)

АНОТАЦІЯ

Бакалаврська робота містить 94 сторінки, 23 рисунків, 4 таблиці, список використаних джерел із 60найменування,

Метою роботи є розробити систематизовану модель стратегій модернізації та інтеграції застарілого коду, яка враховує технічні, організаційні та економічні аспекти та дозволяє ефективно планувати перехід до сучасних архітектур.

Об'єкт дослідження: програмні системи, що містять застарілий код або компоненти, які потребують модернізації чи інтеграції в нові технологічні середовища.

Предмет дослідження: методи, стратегії та інструменти, що застосовуються для модернізації, інтеграції та повторного використання застарілих компонентів, зокрема у контексті сервісно-орієнтованої архітектури (SOA) та сучасних підходів до реінжинірингу.

Результати дослідження: Визначено та систематизовано ключові стратегії модернізації та інтеграції застарілого коду: обгортка, реархітектуризація, повторне використання.

У першому розділі - розкрито контекст проблеми застарілих систем і актуальність їхньої модернізації в цифрову епоху.

Другий розділ - зосереджується на технічних підходах до модернізації, викликах і конкретних методах реінжинірингу..

Третій розділ - досліджує вплив еволюції SOA на підходи до модернізації застарілих систем.

Висновок:. Проведене дослідження засвідчило, що модернізація та інтеграція застарілого коду є критично важливими для забезпечення життєздатності IT-систем в умовах стрімкого розвитку технологій.

КЛЮЧОВІ СЛОВА: ЗАСТАРІЛІ КОМПОНЕНТИ, МОДЕРНІЗАЦІЯ, СЕРВІСНО-ОРІЄНТОВАНА АРХІТЕКТУРА, РЕІНЖИНІРИНГ, ПРОГРАМНІ СИСТЕМИ, ЦИФРОВИЙ КОНТЕНТ, SERVICIFI, ІНТЕГРАЦІЯ

ANNOTATION

The bachelor's thesis contains 94 pages, 23 figures, 4 tables, a list of used sources with 60 titles,

The purpose of the work is to develop a systematized model of strategies for modernization and integration of legacy code, which takes into account technical, organizational and economic aspects and allows you to effectively plan the transition to modern architectures.

Object of research: software systems containing legacy code or components that require modernization or integration into new technological environments.

Subject of research: methods, strategies and tools used for modernization, integration and reuse of legacy components, in particular in the context of service-oriented architecture (SOA) and modern approaches to reengineering.

Research results: Key strategies for modernization and integration of legacy code are identified and systematized: wrapping, re-architecturing, reuse.

The first section - reveals the context of the problem of legacy systems and the relevance of their modernization in the digital age.

The second section - focuses on technical approaches to modernization, challenges and specific methods of reengineering.

The third section - explores the impact of the evolution of SOA on approaches to modernization of legacy systems.

Conclusion:. The study showed that modernization and integration of legacy code are critically important for ensuring the viability of IT systems in the conditions of rapid technological development.

KEYWORDS: LEGACY COMPONENTS, MODERNIZATION, SERVICE-ORIENTED ARCHITECTURE, REENGINEERING, SOFTWARE SYSTEMS, DIGITAL CONTENT, SERVICIFI, INTEGRATION

ЗМІСТ

ВСТУП

Корпоративні інформаційні системи є незамінною опорою для підприємств. Повсякденне ведення бізнесу підприємств стало залежним від інформаційних систем у тому сенсі, що вони можуть збанкрутувати через тривалі системні збої. Корпоративні системи визначаються як впровадження та налаштування пакетів програмного забезпечення, які забезпечують інтеграцію орієнтованих на транзакції даних і бізнес-процесів у всій організації [185]. Такі системи дозволяють підприємству інтегрувати свої дані, які використовуються в усій організації, шляхом бездоганної інтеграції інформації, що проходить через компанію, такої як фінансова та бухгалтерська інформація, інформація про людські ресурси, інформація про ланцюг постачання та інформація про клієнтів [73]. Багато інформаційних систем функціонують на підприємствах протягом десятиліть, і, отже, вони закріпилися на підприємствах. Багатий набір функцій інформаційних систем призводить до того, що багато різних відділів стають залежними від них, що робить їх незамінними. Крім того, за допомогою багатьох налаштувань інформаційні системи були адаптовані відповідно до потреб підприємства, що ще більше зміцнило опору системи в організації. Не дивно, що інформаційні системи були значною мірою адаптовані до потреб підприємства і, отже, включають значні організаційні знання як бізнес-логіку для виконання щоденних операцій.

Актуальність теми. У сучасну цифрову епоху, коли технології стрімко змінюються, значна частина критично важливих систем базується на застарілому коді. Ці системи складно підтримувати, інтегрувати та масштабувати, але вони містять важливу бізнес-логіку. Модернізація та інтеграція таких систем є необхідною умовою для забезпечення їхньої подальшої експлуатації, зменшення витрат на підтримку і підвищення гнучкості. Поява хмарних сервісів, мікросервісної архітектури та підходів DevOps актуалізує питання розробки ефективних стратегій еволюції та повторного використання спадкових компонентів.

Об’єкт дослідження - програмні системи, що містять застарілий код або компоненти, які потребують модернізації чи інтеграції в нові технологічні середовища.

Предмет дослідження - методи, стратегії та інструменти, що застосовуються для модернізації, інтеграції та повторного використання застарілих компонентів, зокрема у контексті сервісно-орієнтованої архітектури (SOA) та сучасних підходів до реінжинірингу.

Мета дослідження - розробити систематизовану модель стратегій модернізації та інтеграції застарілого коду, яка враховує технічні, організаційні та економічні аспекти та дозволяє ефективно планувати перехід до сучасних архітектур.

Завдання дослідження - провести огляд існуючих підходів до модернізації та інтеграції спадщини, визначити технічні виклики, пов’язані з інтеграцією застарілих компонентів, проаналізувати методології, такі як ServiciFi, SMART, SOAMIG., систематизувати стратегії інтеграції (обгортка, рефакторинг, повторне хостування тощо), розробити структуру оцінки ефективності підходів до модернізації, провести аналіз еволюції SOA як цільової архітектури для модернізації.

Наукова новизна - запропоновано інтегровану оцінну модель для планування еволюції застарілих компонентів до сучасної сервісної архітектури.

Методи дослідження - систематичний огляд літератури, порівняльний аналіз методів модернізації, емпіричне дослідження кейсів, архітектурний та функціональний аналіз, методики оцінки технічної та економічної доцільності

Бакалаврська робота містить 94 сторінок, 16 рисунків, три розділи, список використаних джерел із 30 найменуванням.

РОЗДІЛ 1. КОНТЕКСТ ТА ВИМОГИ ДО МОДЕРНІЗАЦІЇ ЗАСТАРІЛОГО КОДУ В ЦИФРОВУ ЕПОХУ

1.1 Огляд проблеми та значення модернізації

Розвиток програмного забезпечення

Термін «еволюція» описує явище, яке стосується прогресивних змін у властивостях і характеристиках класів сутностей, таких як природні види, суспільства, артефакти, теорії. Зміни призначені для збереження статус-кво або покращення придатності в умовах, що змінюються. Такі зміни також неминучі в життєвому циклі програмних систем. Фундаментальна робота Belady & Lehman досі актуальна для розуміння основ еволюції програмного забезпечення. Вони провели емпіричні експерименти на OS/360, щоб зрозуміти еволюцію програмного забезпечення [19]. У цій піонерській роботі з еволюції програмного забезпечення були сформульовані перші три закони еволюції програмного забезпечення, а до 1996 року було сформульовано вісім законів еволюції програмного забезпечення [16]. Усі ці закони визначені в контексті систем E-type [167], тобто систем програмного забезпечення, які вирішують проблему або реалізують комп'ютерну програму в реальному світі і, таким чином, за своєю суттю є більш схильними до змін. Закон безперервних змін – програма Е-типу, яка використовується, повинна постійно адаптуватися, інакше вона стає все менш задовільною [17] – забезпечує внутрішню потребу в еволюції.

Закон зростання складності – коли програма розвивається, її складність зростає, якщо не виконувати роботи для її підтримки або зменшення [17] – визначає потребу в обслуговуванні. Незважаючи на те, що деякі дослідники та практики використовують еволюцію програмного забезпечення як кращу заміну обслуговування [23], це дослідження розрізняє ці два терміни. Обслуговування програмного забезпечення відноситься до дій, які відбуваються в будь-який час після реалізації нового проекту розробки, тоді як

еволюція програмного забезпечення зосереджена на дослідженні поведінки програмних систем, щоб визначити та впровадити зміни для

адаптації/покращення [14]. Крім того, еволюція програмного забезпечення також зосереджується на методах та інструментах, призначених для полегшення еволюції програмного забезпечення після первинної розробки програмного забезпечення [16].

Обслуговування програмного забезпечення включає дії, які в основному спрямовані на підтримку працездатності систем. Swanson [26] класифікував технічне обслуговування на три типи: (i) коригувальне технічне обслуговування – виконується у відповідь на оцінку відмов, (ii) адаптивне технічне обслуговування – виконується в очікуванні змін даних і середовища обробки, і (iii) досконале технічне обслуговування – виконується для усунення ефективності, підвищення продуктивності та покращення ремонтопридатності. Коригуюче та адаптивне технічне обслуговування зосереджено на підтримці систем у працездатному стані, тоді як досконале технічне обслуговування спрямоване на те, щоб підтримувати програмні системи у працездатному стані з меншими витратами та краще задовольняти потреби користувачів. Пізніше Chapin та ін. [54] запровадили четвертий тип обслуговування програмного забезпечення як «профілактичне обслуговування» – виконання, щоб запобігти проблемам у майбутньому шляхом виконання попереджувальних дій. ISO/IEC 14764 використовує їх як чотири категорії обслуговування програмного забезпечення [12].

Застаріла система

Закон постійних змін стверджує, що система повинна постійно адаптуватися, інакше вона стає все менш задовільною. Ідеальне технічне обслуговування вказує на необхідність модифікацій для усунення неефективності, підвищення продуктивності та ремонтопридатності. Крім того, профілактичне технічне обслуговування стосується модифікацій, які виконуються з метою запобігання проблемам до їх виникнення. Закон безперервних змін, досконалого та профілактичного обслуговування робить наголос на адресації (коригування) змін, щоб зробити програмні системи більш придатними для обслуговування, інакше програмні системи поступово перетворюються на застаріле програмне забезпечення – програмні системи, які

значно протистоять модифікаціям і менш придатні для обслуговування. Декілька інших характеристик притаманні застарілим системам, наприклад, негнучкі, крихкі, дорогі в обслуговуванні, відсутність документації, складність розширення та інтеграції з іншими системами та відсутність застарілих експертів. Загалом із застарілими програмними системами часто пов'язані такі проблеми:

- Застарілі системи зазвичай реалізуються з використанням застарілих технологій, можливо, на старих мовах програмування та апаратних платформах.
- Відсутність документації та експертів із застарілих систем призводить до ерозії знань і, отже, до повільного та дорогого процесу обслуговування.
- У застарілих системах часто відсутні чіткі інтерфейси, що вимагає значних зусиль для розширення та інтеграції з іншими системами.

Незважаючи на ці недоліки, притаманні застарілим системам, важливість застарілої системи широко визнається. Застарілі системи є основою підприємств і часто розглядаються як організаційний актив із високою економічною цінністю [285]. Ці системи є критично важливими та включають багато бізнес-логіки, яка є результатом багатьох років кодування, розробок, удосконалень, модифікацій і тестування. Тому підприємства не можуть просто відмовитися від своїх застарілих систем, незважаючи на кошмари обслуговування та основні проблеми ерозії програмного забезпечення [21].

Модернізація програмного забезпечення

Підприємства з успадкованими системами стикаються з дилемою [22]. Незважаючи на невиправдані витрати на технічне обслуговування, підприємства не можуть просто позбутися застарілих програмних систем, оскільки вони є основними системами для ведення повсякденного бізнесу.

Таким чином, зростає імпульс для розвитку та повторного використання цих застарілих систем у нових технологічних середовищах за допомогою модернізації програмного забезпечення [32]. Основною метою модернізації програмного забезпечення є зниження витрат на обслуговування та підвищення гнучкості. Визначення модернізації програмного забезпечення – Ми визначаємо

застарілу модернізацію як «процес розвитку існуючих систем програмного забезпечення шляхом заміни, повторної розробки, повторного використання або міграції програмних компонентів і платформ, коли традиційні практики обслуговування більше не можуть досягти бажаних властивостей системи» [14]. Через проблеми, властиві застарілим програмним системам, були запропоновані різні методи модернізації програмного забезпечення. Такі методи модернізації можна загалом класифікувати як стратегії модернізації програмного забезпечення.

Стратегії модернізації програмного забезпечення

Модернізацію програмного забезпечення можна розділити на чотири різні стратегії [8], які коротко обговорюються нижче:

1. Стратегія заміни – це спосіб виведення з експлуатації застарілої системи та заміни її на комерційний пакет (COTS). Заміна вважається менш ризикованою, але можливість повторного використання існуючої бізнес-логіки, вбудованої в застарілу систему, невелика або взагалі відсутня. Зазвичай застарілі програмні системи модифікуються та налаштовуються протягом життєвого циклу та є джерелами недокументованої бізнес-логіки або корпоративних знань. Існує варіант, коли пакет COTS змінюється відповідно до потреб підприємства, але це вимагає значних витрат. Крім того, немає гарантії, що замінена нова система буде такою ж надійною та функціональною, як оригінальна [8].

2. Обгортка – це одна з найбільш широко використовуваних стратегій модернізації, яка дозволяє інкапсулювати існуюче застаріле програмне забезпечення для повторного використання в новій цільовій архітектурі [25]. Загалом упаковка забезпечує налаштований доступ до застарілого коду з мінімальними змінами в самій кодовій базі, так що упакований компонент може використовуватися іншими компонентами програмного забезпечення. Обертання — це стратегія швидкого виграшу, і її можна використовувати, коли застаріла система має високу бізнес-цінність. Однак упаковка не зменшує витрати на обслуговування, скоріше збільшує їх, оскільки підприємство також має підтримувати рівень інтерфейсу (обгортки).

3. Реконструкція – це стратегія переробки застарілих функціональних можливостей системи. Однак ризик невдачі зазвичай занадто великий для того, щоб підприємства могли серйозно прийняти підхід до реконструкції [3], а керівництво не бажає витрачати значну суму інвестицій на підхід, який має величезний ризик невдачі. Що стосується вартості, реконструкція дійсно спричиняє значні витрати на розробку та обмежене повторне використання існуючих успадкованих активів. Менше повторне використання наявних активів виправдано тим фактом, що документація застарілих програмних систем загалом не є актуальною.

4. Міграція – стосується трансформації застарілих програмних систем у новий технологічний контекст шляхом максимального повторного використання [127]. Порівняно з іншими стратегіями, стратегія міграції є дорогою та займає багато часу. Однак стратегія міграції поступово дозволяє внутрішню реструктуризацію, повторне використання та модифікацію застарілих систем у нову цільову систему, тим самим потенційно знижуючи витрати на обслуговування, пов’язані із застарілими системами в довгостроковій перспективі [27].

На рисунку 1.1 показано порівняння цих стратегій щодо вартості та повторного використання існуючих активів. Кожна стратегія модернізації має свої плюси та мінуси, тому низка факторів, таких як доступний бюджет, ресурси, часові обмеження, відіграють важливу роль у виборі стратегії. Часто дві або більше стратегій модернізації об’єднують для проведення застарілої модернізації, оскільки немає жодного засобу для вирішення проблеми [8].

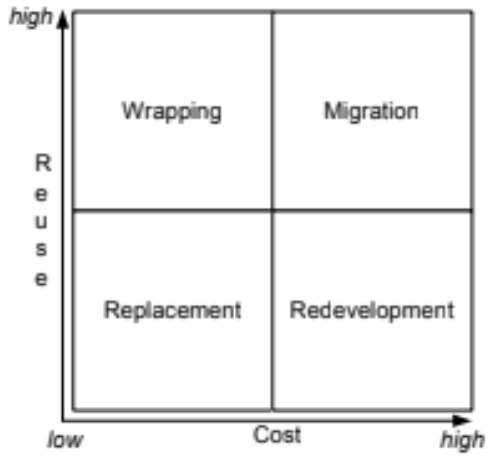

Рисунок 1.1 - Порівняння стратегій модернізації щодо вартості та повторного використання [8]

Методи модернізації програмного забезпечення

За останні чотири десятиліття було описано безліч методів модернізації застарілих систем. Sneed [25, 26] представляє метод міграції застарілих систем програмного забезпечення з мейнфрейму на архітектуру клієнт-сервер за допомогою методів упаковки. Sneed & Majnar [29] обговорюють використання упаковки для міграції застарілих систем програмного забезпечення до архітектури клієнт-сервер на різних рівнях інкапсуляції, таких як завдання, транзакція, програма, модуль і процедура. Souder & Mancoridi [26] представляють інструмент для інтегрувати застарілу програмну систему в розподілене середовище за допомогою технології упаковки. Канфора та ін. [4] описують підхід до декомпозиції застарілих програмних систем за допомогою нарізки програм, а потім використання обгорток для переходу на архітектуру клієнт-сервер. Для міграції застарілих програмних систем на архітектуру клієнт-сервер переважає техніка обгортання [14].

З появою об’єктно-орієнтованої (ОО) парадигми повідомляється про численні успадковані методи модернізації об’єктно-орієнтованої мови програмування. Demeyer та ін. [7] повідомляють про декілька шаблонів об’єктно-орієнтованої реінжинірингу для модернізації об’єктно-орієнтованих

мов. Lucia та ін. [7] представляють шість етапів послідовного переходу до об'єктно-орієнтованого процесу міграції, який охоплює зворотне проектування та реінжиніринг. Cimitile та ін. [4] пропонують метод декомпозиції застарілих систем на об'єкти за допомогою зворотного проектування. Sneed [24, 25] повідомляє про методи модернізації для перенесення програми COBOL у функціонально еквівалентну програму ОО. Newcomb [203] описує інструмент реінжинірингу, який автоматично перетворює процедурну програму в функціонально порівнянну ООП-систему. Zou & Kontogiannis [30] розробляють верстак реінжинірингу, який не тільки переносить процедурну мову в ОО, але також дозволяє моделювати вимоги до якості для цільової системи мігрантів.

В останнє десятиліття розвиток веб-технологій сприяв модернізації застарілої системи програмного забезпечення (наприклад, [24, 14, 4, 5]). Зокрема, сервіс-орієнтована архітектура (SOA) була популярною цільовою архітектурою для модернізації застарілої системи програмного забезпечення, і були запропоновані різні підходи до модернізації застарілої SOA (наприклад, див. систематичні огляди літератури Razavian & Lago [22] і Almonaies et al. [8]). Останнім часом хмарні обчислення розглядаються як нова цільова платформа для модернізації застарілого програмного забезпечення. Систематичний огляд літератури (SLR) Jamshidi et al. [19] повідомили, що з 2010 по 2013 рік було зареєстровано 23 різні підходи до модернізації із застарілих до хмарних обчислень. Популярність модернізації із застарілих до хмарних обчислень відображається в ряді дослідницьких проектів, таких як REMICS [2], ХУДОЖНИК [26]6, MODAClouds [10].

Техніки, що використовуються для модернізації, різноманітні, і переважно переважає технологія обгортання. Інші часто використовувані методи включають розрізання програм [14, 16, 5, 18, 30], моделювання функцій [17, 18], методи трансформації моделі [5, 9, 12], оболонки [26, 28, 22, 24, 28], аспекти [13] та мови архітектурних шаблонів [10, 11]. Дослідження модернізації застарілих систем, виявлені в академічних колах, здебільшого орієнтовані на технології. Вони надають різні техніки/методи для полегшення

модернізації застарілої системи та вказують на різні проблеми, з якими стикаються під час застосування цих технік/методів. Спостерігається недостатня увага приділяється бізнес-питанням модернізації застарілого програмного забезпечення [19, 20].

Сервісно-орієнтована архітектура

Сервісно-орієнтована архітектура (SOA) [3] — це архітектурна парадигма, яка представляє відкриту, розширювану та комбіновану архітектуру програмного забезпечення, побудовану з повторно використовуваних компонентів програмного забезпечення, відомих як служби. SOA фокусується на повторному використанні компонентів шляхом відділення інтерфейсу від внутрішньої реалізації. Основні принципи, які просувають SOA, включають слабий зв'язок, абстракцію основної логіки, гнучкість, гнучкість, багаторазове використання, автономність, відсутність стану та можливість виявлення [83, 206]. З точки зору модернізації програмного забезпечення, SOA обіцяє повторно використовувати існуючі застарілі активи шляхом інкапсуляції їх як додаткових послуг [27, 22]. Channabasavaiah & Holley [2] стверджують, що підприємства отримають наступні переваги, модернізувавши застарілі системи програмного забезпечення до SOA:

Використовуйте наявні активи: Однією з ключових і найважливіших переваг SOA є повторне використання наявних застарілих активів шляхом інкапсуляції застарілих функцій як додаткових послуг. Повторне використання застарілих активів дозволяє підприємству зберегти бізнес-цінність інвестицій, зроблених у розробку застарілих програмних систем протягом багатьох років. Крім того, сервіси приховують реалізацію та складність платформи застарілої програми та забезпечують уніфікований механізм доступу через сервісні інтерфейси.

- Швидший час виходу на ринок: SOA полегшує створення переліку послуг, який містить як ділові, так і технічні деталі бізнес-послуг. Підприємства можуть використовувати перелік послуг, щоб визначити послуги, які відповідають їхнім вимогам, і використовувати їх повторно, коли це можливо.

Крім того, завдяки повторному використанню існуючих застарілих активів час, необхідний для проектування, розробки, тестування та розгортання, значно скорочується, таким чином скорочуючи час виходу нових продуктів на ринок.

- Зниження витрат: У середовищі SOA кілька існуючих сервісів можуть бути скомпоновані для надання додаткового сервісу за допомогою композиції сервісів [28]. Коли з'являються нові вимоги до нових продуктів, підприємства можуть використовувати перелік послуг, щоб визначити відповідні послуги та скласти їх для надання нової послуги. Водночас повторне використання існуючих активів значно сприяє зниженню витрат порівняно з новою розробкою продуктів.
- Гнучкість: Платформа обчислення та розробки може значно відрізнятися в межах підприємства, що призводить до проблем сумісності [28]. SOA може значно зменшити проблеми сумісності, приховуючи складність платформи за допомогою визначення стандартних сервісних інтерфейсів. Інкапсулюючи обчислювальну техніку та складність платформи, SOA відкриває широкі можливості для інтеграції роз'ємних програм на підприємствах.

У цій роботіSOA розглядається як цільова архітектура застарілої модернізації. Зокрема, частина I роботіприсвячена модернізації застарілих програмних систем до середовища SOA.

1.2 Стратегії інтеграції застарілих компонентів

Застарілі програмні системи, їх характеристики та можливі рішення для модернізації цих систем є добре відомими та давно дослідженими областями наукової спільноти програмної інженерії. Пошукова робота, проведена наприкінці 70-х і на початку 80-х щодо еволюції програмного забезпечення та обслуговування програмного забезпечення, є новаторськими роботами, які все ще актуальні для застарілих програмних систем та їх модернізації. Зокрема, закони еволюції програмного забезпечення Лемана [16] і внесок Суонсона в підтримку програмного забезпечення [26] все ще актуальні для розуміння

основи еволюції програмного забезпечення. На основі концепції еволюції програмного забезпечення та обслуговування програмного забезпечення, було повідомлено про багато досліджень у сфері застарілих програмних систем та їх модернізації. Один із ключових емпіричних внесків у модернізацію застарілого програмного забезпечення належить Сніду за його роботу з 1984 року щодо оновлення програмного забезпечення [25]. У середині 90-х широко обговорюється термін «успадковані системи» та їхні характеристики, підкреслюючи, зокрема, високу вартість обслуговування цих систем. Приблизні оцінки витрат, виділених на обслуговування програмного забезпечення, можуть досягати 70-80% від загальної вартості життєвого циклу системи [18]. Важливість еволюції та підтримки програмного забезпечення було чітко видно під час проблеми «2000 рік (Y2K)» [8]. Сміт та ін. [24] повідомили, що однією з причин серйозності проблеми Y2K є існування застарілих програмних систем та їх величезна кодова база.

З тих пір застарілі програмні системи та їх модернізація постійно набували значення і перемістилися в центр уваги інженерів програмного забезпечення [23, 21]. Протягом багатьох років спільнота розробників програмного забезпечення уважно стежила за розвитком програмного забезпечення. Ще в 1996 році платформи клієнт-сервер використовувалися як цілі для модернізації застарілих програмних систем (наприклад, Sneed [25] і Canfora et al. [4). Пізніше, з розвитком Інтернет-технологій, тенденція модернізації застарілого програмного забезпечення набрала обертів для модернізації до веб-додатків (наприклад, Струліа та ін. [26], Лавері та ін. [16] і Де Люсія та ін. [7]). З обіцяними перевагами SOA модернізація застарілого програмного забезпечення стала в основному зосереджена на переході до SOA (наприклад, SLR від Razavian & Lago [22], Almonaies et al. [8]). Останнім часом, завдяки потенціалу, який пропонують хмари, тепер модернізація застарілого програмного забезпечення звернулася до модернізації до хмар (наприклад, застаріле до хмар SLR Jamshidi та ін. [12].

Зміни програмних систем протягом їхнього життєвого циклу є

безперервним процесом. Такі зміни спричинені мінливими вимогами, технологіями та вимогами ринку. Протягом багатьох років зворотне проектування відігравало ключову роль у модернізації застарілої системи програмного забезпечення. Мюллер та ін. [19] стверджують, що методи зворотного проектування були ключовими в допомозі модернізації застарілої системи програмного забезпечення. Chikofsky & Cross [6] визначають зворотне проектування як «процес аналізу системи програмного забезпечення для ідентифікації компонентів системи та їхніх взаємозв'язків і створення представлень системи в іншій формі або на вищому рівні абстракції». Кілька методів зворотного проектування широко використовуються для допомоги в модернізації застарілих програмних систем. Наприклад, такі методи розуміння системи, як розташування функцій, візуалізація програми, аналіз вихідного коду, нарізка програми, аналіз концепції, відновлення архітектури програмного забезпечення, широко використовуються для розуміння та допомоги в модернізації застарілих програмних систем.

Про застаріле програмне забезпечення та його модернізацію регулярно повідомляють на конференціях із програмного забезпечення найвищого рівня, таких як Міжнародна конференція з розробки програмного забезпечення8(ICSE) та Основи розробки програмного забезпечення9(FSE) під заголовками еволюції програмного забезпечення та обслуговування програмного забезпечення, що вказує на активне поле досліджень. На додаток до цього є дві спеціальні та провідні конференції – Міжнародна конференція з обслуговування та розвитку програмного забезпечення (ICSME – раніше відома як Міжнародна конференція з обслуговування програмного забезпечення).10 (ICSM)) та Міжнародна конференція з аналізу програмного забезпечення, еволюції та реінжинірингу (SANER– об'єднання Європейської конференції з обслуговування програмного забезпечення і Реінжиніринг11 (CMSR) та Робоча конференція з зворотного проектування12 (WCRE) – де дослідники публікують свої дослідження про застарілу програмну систему та її модернізацію. Місця проведення семінарів, такі як симпозіум з обслуговування та розвитку сервіс-

орієнтованих систем і хмарних середовищ13 (MESOCA), Міжнародний семінар з принципів розвитку програмного забезпечення14 (IW PSE) регулярно публікує нові дослідницькі ідеї та звіти про досвід із галузі, пов'язаної з доменом застарілих програмних систем.

Застарілі програмні системи та їх модернізація також регулярно викликають значний інтерес у промисловості. Фірми з дослідження ринку, такі як Gartner, Forrester Research, регулярно зазначають, що модернізація застарілого програмного забезпечення є одним із головних пріоритетів у галузі. Крім того, більшість фінансових установ все ще використовують свої застарілі системи програмного забезпечення в бек-офісі для виконання своїх повсякденних операцій.

Дослідження, про які йдеться в цій дисертації, актуальні як для наукового співтовариства, так і для промисловості. Для наукового співтовариства це дослідження додає знання про еволюцію програмного забезпечення та підтримку програмного забезпечення, надаючи структурований метод модернізації застарілої системи програмного забезпечення. Це дослідження також дає зрозуміти, як промисловість цінує свої застарілі системи та з якими проблемами стикається галузь під час модернізації. Ці виклики можна розглядати як напрямок майбутніх досліджень для наукової спільноти. Нарешті, ми віримо, що це дослідження сприятиме передачі технологій і знань між академічними та промисловими колами в області модернізації застарілих програмних систем.

1.3 Дослідницький підхід

У цьому підрозділі ми обговорюємо підхід до наукового дослідження та методи дослідження, використані в цій дисертації. Ми використовуємо поняття практичної проблеми та проблеми знання в контексті науки про дизайн [29] (див. рис. 1.2), щоб обґрунтувати цілі дослідження.

Wieringa [29] визначає практичну проблему як різницю між тим, як світ сприймають зацікавлені особи, і тим, яким вони хотіли б, щоб це було.

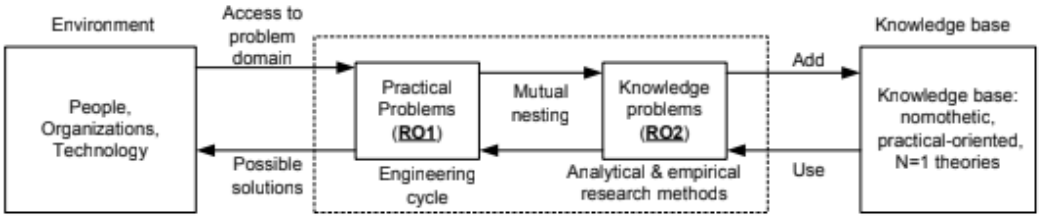

Рисунок 1.2 - Структура науки про дизайн

Практичні проблеми вимагають рішень, які вносять зміни у світ зацікавлених сторін для досягнення цілей. Навпаки, проблема знання — це різниця між поточними знаннями зацікавленої сторони про світ і тим, що вони хотіли б знати. На відміну від практичних проблем, проблеми знання вимагають не зміни світу, а зміни знання про світ. RO1 – це практична проблема, тоді як RO2 – це проблема знання. Важливо зазначити, що практична проблема, якщо її розкласти на підпроблеми, може включати запитання щодо знань – щоб полегшити розуміння поточного стану ключової проблеми або щоб визначити, чи артефакти відповідають цілям. Приклад декомпозиції практичної задачі зображено на рисунку 1.3.

Ціль дослідження 1 (RO1) – «Розробити метод модернізації програмного забезпечення, який включає технічні та бізнес-аспекти». У RO1 ми досліджуємо існуючі академічні методи модернізації застарілої системи програмного забезпечення в контексті модернізації до сервісно-орієнтованої архітектури (SOA) з метою розробки методу модернізації застарілого до SOA, який вирішує технічні та бізнес-проблеми модернізації. У рамках RO1 метод є рішенням, яке змінює світ відповідної зацікавленої сторони.

Логічною структурою для розв’язання практичної проблеми є інженерний цикл – підхід, який починається з дослідження для розуміння проблеми, потім уточнюється та реалізується проект рішення з його перевіркою [29, 29]. Щоб дослідити RO1, ми розробляємо метод модернізації застарілої SOA та перевіряємо метод. Ми сформулювали наступне дослідницьке питання для дослідження RO1.

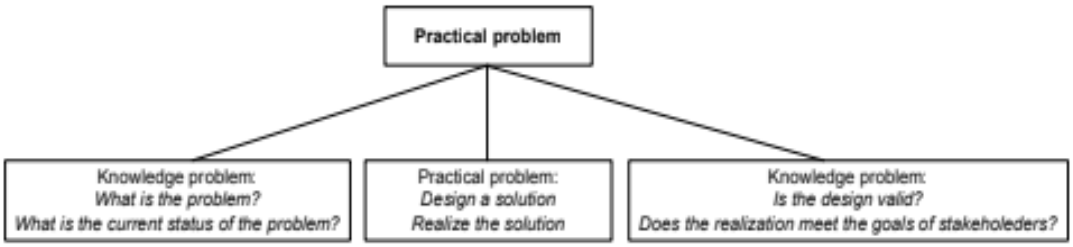

Рисунок 1.3: Декомпозиція практичної проблеми (адаптовано з Wieringa

ЗП 1. Яким чином можна розробити процес модернізації, який би полегшив підприємствам модернізацію систем програмного забезпечення? Це питання дослідження представляє практичну проблему в області модернізації застарілого програмного забезпечення. Як Wieringa [29], ми використовуємо інженерний цикл для дослідження проблеми, а потім розробляємо та перевіряємо структурований метод. Метод зосереджений на модернізації успадкованої SOA, у якій ми об'єднуємо технічні та бізнес-проблеми, пов'язані з модернізацією успадкованої SOA. Це дослідницьке питання далі поділяється на три підпитання:

ЗП 1.1 Які (суттєві) кроки для поєднання бізнес-аспектів і технічних аспектів у модернізації програмного забезпечення? У літературі повідомлялося про кілька підходів до модернізації застарілих систем програмного забезпечення для модернізації застарілих систем до нової технології. Значна кількість таких підходів зосереджена на розробці допоміжної технології для вирішення перспективи технічної модернізації (тобто методи впровадження для повторного використання застарілої системи програмного забезпечення). Інші підходи зосереджені на розробці стратегії модернізації для визначення

доцільності міграції. Крім того, недостатньо уваги приділяється бізнес-аспектам модернізації незважаючи на те, що дослідники [17, 14, 20] стверджують необхідність консолідованого методу, який поєднує в собі всі вищезазначені аспекти. Щоб вирішити цю практичну проблему та відповісти на питання дослідження, ми визначаємо кроки, необхідні для поєднання бізнес- і технічних аспектів модернізації програмного забезпечення. Для консолідації методу ми використовуємо інженерію методів [3] – метод розробки

інформаційної системи для створення розширених методів розробки шляхом повторного використання частин існуючих методів. Консолідований успадкований метод SOA перевірено експертами з модернізації та додатково підтверджено двома прикладами.

RQ 1.2 Який сучасний стан модернізації програмного забезпечення в академічних колах? Після того, як ми продемонстрували доцільність об'єднання різних аспектів в одному методі модернізації застарілої версії, ми систематично досліджуємо методи та методи, про які повідомляють наукові кола щодо модернізації програмного забезпечення із застарілої версії на SOA. Ми використовуємо підхід систематичного огляду літератури (SLR), щоб відповісти на цю проблему знань. SLR використовується для створення переліку методів і технік, що використовуються на різних етапах модернізації програмного забезпечення від старого до SOA. З цією метою було визначено та оцінено 121 дослідницьку статтю, щоб створити перелік поточних дослідницьких підходів, методів, інструментів і технік. Щоб мінімізувати упередженість дослідників, ми ретельно дотримувалися вказівок, викладених Kitchenham та ін. [14] для виконання SLR. На кожному кроці процесу SLR ми виявили потенційні загрози дійсності та вжили відповідних заходів для пом'якшення цих загроз дійсності. Наприклад, щоб мінімізувати упередженість дослідника, 121 статтю було розподілено між 5 дослідниками для визначення методів, а пізніше результати були перехресно перевірені дослідником, відмінним від того, хто класифікував методи спочатку. Крім того, критерії включення та виключення для документів були чітко задокументовані, щоб мінімізувати відсутність відповідних досліджень. використовується для модернізації застарілої системи програмного забезпечення.

RQ 1.3 Яким чином можна розробити процес модернізації програмного забезпечення для SOA на основі існуючих методів і технік? Це практична проблема з метою розробки процесу модернізації застарілої версії SOA на основі існуючих методів модернізації. Це дослідницьке питання розширює етапи RQ 1.1 і спрямоване на розробку поетапно структурованого методу для

модернізації застарілої версії SOA. Для кожного етапу ми представляємо обґрунтування, щоб обґрунтувати його необхідність, поточну практику та проблеми, які потребують додаткової уваги. Це дослідження ґрунтується на обґрунтуванні, що існує потреба в методі модернізації структурованої спадщини SOA, який включає не лише технічні питання, але й бізнес-питання [17, 14, 20]. Потім запропонований структурований процес оцінюється шляхом перенесення функцій двох простих, але репрезентативних програм до SOA. Для подальшої перевірки структурованого процесу ми вибрали 17 наукових статей, у яких розповідається про спадщину модернізації SOA з 2000 по 2011 рік, і зіставили дії, описані в них, із фазами структурованого процесу.

Ціль дослідження 2 (RO2) – «Визначити, як модернізація програмного забезпечення сприймається та проводиться на практиці». У RO2 ми досліджуємо, як у промисловості сприймаються застарілі системи програмного забезпечення та модернізація застарілого програмного забезпечення. Ця проблема знань спрямована на надання відповідей щодо зміни або оновлення поточних знань про застарілі системи та модернізацію (застарілого) програмного забезпечення. Ми вважаємо, що ці знання мають значну цінність через те, що існує розрив у знаннях між академічними та промисловими колами щодо розуміння застарілих систем та їх модернізації, що перешкоджає передачі знань. Наприклад, Razavian & Lago [22] вказують на те, що 97% підходів до модернізації SOA від академічної спадщини не відповідають промисловим цілям у контексті модернізації від спадщини до SOA [23]. Ми віримо, що знання, отримані від RO2, можуть бути використані для впровадження

академічних методів модернізації в промисловості. Щоб дослідити цю мету дослідження, ми використовуємо різні емпіричні методи дослідження: (i) тематичні дослідження, щоб дослідити, як модернізація застарілої системи виконується в промислових масштабах, (ii) метод обґрунтованої теорії, щоб визначити, як практики бачать застарілі системи та з якими проблемами вони

стикаються під час модернізації, і (iii) змішані методи, такі як поєднання тематичних досліджень та інтерв'ю для розуміння впливу застарілої

модернізації. Ми сформулювали наступне дослідницьке питання для дослідження RO2.

ЗП 2 Яке уявлення практиків про модернізацію програмного забезпечення? У RQ 2 ми досліджуємо промислове сприйняття застарілих систем і модернізації застарілого програмного забезпечення. Зокрема, ми досліджуємо, які характеристики застарілих програмних систем все ще забезпечують їх працездатність, які ключові чинники модернізації, з якими ключовими проблемами стикаються в процесі модернізації та які бізнес-цілі досягаються після проведення модернізації застарілого програмного забезпечення. Щоб отримати знання з таких питань, ми далі розділяємо досліджуване питання на три підпитання:

ЗП 2.1 Як на практиці виконується масштабна модернізація програмного забезпечення? Щоб створити контекст для модернізації застарілих систем у промисловості, у цьому дослідницькому питанні досліджується, як на практиці модернізуються застарілі системи програмного забезпечення. Далі досліджується, які методи використовуються для модернізації та з якими проблемами стикаються під час модернізації. Очевидно, що існує відносно небагато прикладів модернізації промислового застарілого програмного забезпечення, про які повідомляється в наукових колах, що обмежує знання про те, як модернізація проводиться в промислових масштабах. Отже, це дослідницьке питання представляє широкомасштабний метод модернізації застарілого програмного забезпечення для SOA у фінансовій сфері та детально описує процес модернізації. Для проведення цього дослідження

використовується єдиний метод дослідження конкретного випадку [29]. Одним із обмежень окремого дослідження є можлива упередженість у зборі та інтерпретації даних. Це потенційне упередження в цьому дослідженні зведено до мінімуму завдяки використанню кількох методів збору даних (документація, інтерв'ю, семінари) і залученню двох дослідників до проекту для регулярної

перехресної перевірки результатів.

ЗП 2.2 Які розбіжності між сприйняттям застарілого програмного

забезпечення та його модернізацією в наукових колах і промисловості? Попередні висновки дослідницького питання RQ 2.1 і широке використання застарілих програмних систем у фінансовій сфері спонукають нас дослідити це дослідницьке питання, у якому ми прагнемо визначити, як застарілі системи та їх модернізація сприймаються в промисловості. Зокрема, які характеристики застарілих програмних систем забезпечують їх працездатність у промисловості, які рушійні сили призводять до модернізації та які проблеми постають під час модернізації? Ми використовуємо метод дослідження обґрунтованої теорії – дослідницький метод дослідження, який спрямований на відкриття нових перспектив і уявлень, а не на підтвердження існуючих для дослідження цього дослідницького питання [103], і аналізуємо інтерв'ю 26 практиків. Щоб перевірити результати обґрунтованого дослідження теорії, ми використовуємо опитування як метод тріангуляції даних – процес перевірки, який використовує більше одного джерела даних, щоб підвищити (зменшити) впевненість у висновку шляхом надання підтверджуючих (суперечливих) доказів. Результати інтерв'ю відповідають опитуванню.

ЗП 2.3 Як часто досягаються бізнес-цілі перед модернізацією після «технічно» успішної модернізації програмного забезпечення? У цьому дослідницькому питанні ми досліджуємо, що означає «успішна» модернізація застарілої системи з точки зору бізнесу. На даний момент модернізація застарілого програмного забезпечення вважається успішною після завершення технічної модернізації. Проте дослідження результатів постмодернізації були обмежені. За допомогою цього дослідницького питання ми прагнемо визначити, яких бізнес-цілей досягають підприємства після модернізації своїх застарілих програмних систем. Ми досліджуємо п'ять компаній, які завершили модернізацію програмного забезпечення, і досліджуємо бізнес-цілі, які стоять за модернізацією. Далі ми визначаємо, яких із цих бізнес-цілей досягає модернізація програмного забезпечення. Ми використовуємо кілька тематичних досліджень [297], щоб дослідити це питання дослідження та використовуємо кілька джерел даних, щоб мінімізувати упередженість дослідження.

Методи дослідження

У цій роботіми використали ряд методів дослідження, які переважно використовуються в розробці програмного забезпечення та дослідженнях інформаційних систем. Ми обговорюємо методи дослідження в наступних підрозділах:

Дослідження науки про дизайн [12] спрямоване на створення та оцінку артефактів IT, призначених для вирішення ідентифікованих організаційних проблем. Потім такі артефакти розробляються та перевіряються в координації з базою знань. На рисунку 1.4 зображено наукове дослідження дизайну, засноване на Hevner et al. [11]. У Частині I цієї роботіми розробляємо метод модернізації спадщини SOA і згодом перевіряємо цей метод. Цей контекст розробки та оцінки методу модернізації спадщини SOA належним чином охоплюється методом наукового дослідження проектування. Наука про дизайн дозволяє поєднувати знання з існуючих теорій і методів (наприклад, існуючі методи модернізації, відома академічна література) з новою 15 дані (наприклад, уроки, отримані з тематичних досліджень) для отримання нових результатів і теорій. Потім такі результати оцінюються за допомогою інших джерел даних, таких як інтерв'ю, контрольні випадки або опитування [22].

Дослідження IC

Дослідження в галузі інформаційної науки (IS) зазвичай є комбінацією наук про поведінку та науки про дизайн [21]. Поведінкова наука прагне розробити та обґрунтувати теорії навколо інформаційних систем, тоді як наука про дизайн спрямована на створення інновацій навколо інформаційних систем для підвищення ефективності та ефективності [78, 11]. Це дослідження поєднало обидва (науку про дизайн і науку про поведінку) і призвело до розробки та оцінки кількох артефактів дизайну, таких як метод модернізації застарілої системи програмного забезпечення для покращення процесу модернізації. Етап оцінювання має на меті обґрунтувати придатність методу шляхом перевірки його за допомогою експертів або тематичних досліджень.

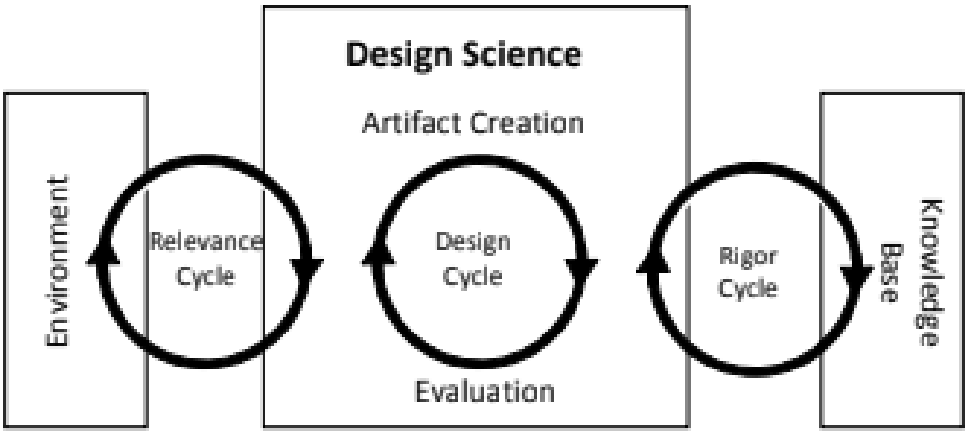

Рисунок 1.4 - Метод наукового дослідження дизайну

Навколишнє середовище

Середовище визначає проблемний простір, у якому знаходяться явища, що цікавлять, і в якому проводиться дослідження. Простір складається з практиків, бізнес-організацій та їхніх існуючих і запланованих технологій і практик. Крім того, середовище включає цілі, завдання, проблеми та можливості, які визначають потреби бізнесу, як вони сприймаються людьми всередині організації [12]. Що стосується навколишнього середовища, це дослідження особливо зосереджено на дослідниках модернізації застарілих програмних систем, промислових практиках, а також на різних існуючих процесах модернізації, методах, інструментах і техніках підтримки модернізації, але не обмежується цим.

База знань

База знань забезпечує наукову основу (наприклад, теорію та літературу) і методології (наприклад, системи методів, що використовуються в певній галузі дослідження інформаційних систем), з яких і через які виконуються дослідження в галузі інформатики [11]. Це дослідження пов’язане із застарілими програмними системами та процесом модернізації, що, нарешті, робить внесок у існуючу базу знань спільноти еволюції та підтримки програмного забезпечення. Результати та висновки цього дослідження важливі для практиків. Крім того, внесок науки про поведінку та науки про дизайн у

дослідження інформаційних систем застосовується до потреб бізнесу в конкретному середовищі та доповнює зміст бази знань для подальших досліджень і практики [12].

Оцінка артефактів, розроблених у рамках дослідження ІБ, є безперервним процесом, який називається циклом проектування. У цьому безперервному процесі існує постійна петля зворотного зв'язку з навколишнім середовищем і База знань. Ці цикли зворотного зв'язку відомі як цикл релевантності та цикл строгості відповідно [12, 21]. Метод наукового дослідження дизайну використовується в цій роботідля розробки кількох артефактів дизайну, таких як методи модернізації, створення теорії для їх оцінки.

Приклади

Суттєві дослідження в рамках цієї роботіспираються на спостереження за модернізацією застарілої промислової системи програмного забезпечення в реальному світі. Отже, важливо вибрати відповідний метод дослідження в інженерії програмного забезпечення, який дозволяє нам вивчати сучасні явища в їх природному контексті [23]. Метод дослідження прикладів є відповідним методом дослідження для спостереження за процесами модернізації застарілої системи в реальному світі.

Метод дослідження конкретного випадку використовується в багатьох ситуаціях, щоб зробити внесок у наші знання про окремі чи організаційні явища [29]. Тематичні дослідження прагнуть зобразити, як це бути в конкретній ситуації, розглядаючи випадок або явище в контексті реального життя, зазвичай використовуючи багато типів даних. Дослідження кейсів передбачає уважне вивчення людей, тем, проблем або програм з метою розуміння, а також створення теорії та тестування [29]. На рисунку 1.5 зображено різні типи розробки тематичних досліджень, як показано Інь [29]. На рисунку 1.5 (A) представлено цілісний дизайн окремого випадку, який передбачає інтенсивний опис та аналіз окремого випадку. Рисунок 1.5 (B) ілюструє дизайн кількох випадків, у якому аналізуються кілька джерел/випадків.

У сфері інформаційних систем успішне завершення тематичного дослідження вимагає ініціативи, прагматизму, здатності використовувати

несподівані можливості, а також оптимізму та наполегливості перед обличчям труднощів і несподіваних подій, особливо під час збору даних. Крім того, часто повідомляється, що дослідження, засновані на тематичних дослідженнях, є занадто конкретними та спрямованими на створення гіпотез, що призводить до загроз достовірності та упередженості дослідження [90]. У цьому дослідженні ми пом'якшуємо ці проблеми, визначаючи репрезентативні випадки, розробляючи та дотримуючись протоколу аналізу випадків. Крім того, у наших кейсах ми суворо дотримувалися вказівок щодо створення протоколів кейсів, як це передбачено Jansen & Brinkkemper [13] та Yin [29].

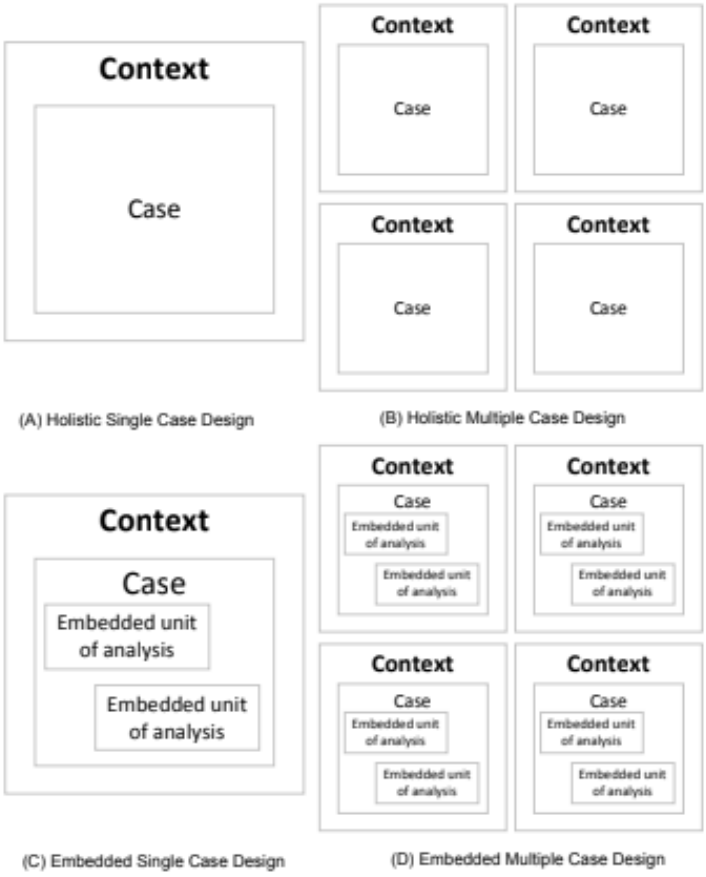

Рисунок 1.5 - Типи проектів тематичних досліджень, адаптованих з Інь

За останні чотири десятиліття повідомлялося про безліч методів дослідження в контексті модернізації застарілої системи програмного забезпечення, цільовою архітектурою якої є SOA. Однак немає систематичного огляду цього дослідження, зокрема технік, методів і підходів, які

використовуються для розвитку застарілих систем до середовища SOA. У систематичному огляді, проведеному Razavian & Lago [22], обговорюється класифікація міграції SOA на вісім сімейств. Однак у цьому огляді не міститься перелік методів і прийомів, які використовуються для модернізації застарілої SOA. Тому ми застосували метод систематичного огляду літератури (SLR), щоб систематично збирати й аналізувати існуючу літературу та надати широкий перелік використаних методів і технік. SLR — це підхід, що ґрунтується на

доказах і спрямований на надання відповідей на деякі дослідницькі питання шляхом документування вичерпного підсумку поточної літератури, аналізу та синтезу результатів [15]. SLR є корисним і потужним дослідницьким методом для збору та аналізу наявної роботи, що є звичайним завданням для встановлення базових знань для будь-якого дослідження.

Однак SLR не є єдиним методом дослідження для систематичного збору та аналізу існуючої літератури. Систематичне картографування (SMS) [15] також є потенційним підходом. Проте Кітченхем [15] стверджує, що SMS підходить, коли на певну тему існує небагато статей, або тема надто широка чи розрізнена. Зокрема, SMS більше спрямована на розкриття дослідницьких тенденцій, а не на надання відповідей на конкретні дослідницькі питання [25]. У нашому випадку SLR вважається більш доцільним через те, що опубліковано велику кількість досліджень, і мета полягає в тому, щоб надати повний огляд дослідницької області на основі всіх опублікованих документів про модернізацію спадщини до SOA. Типовий SLR складається з трьох етапів, як показано на малюнку 1.6, з деталями дій, які виконуються в межах кожного етапу. Спочатку фаза перегляду плану забезпечує контекст огляду шляхом визначення дослідницьких питань і розробки протоколу огляду. Етап проведення огляду являє собою операціоналізацію процесу огляду шляхом

вибору джерел даних, визначення пошукових запитів, вибору первинних досліджень (часто на основі сканованої теми та анотації) та аналізу даних. Нарешті, результати перевірки документуються у звіті.

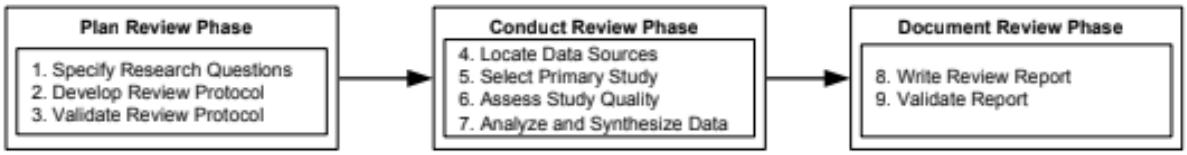

Рисунок 1.6 - Процес SLR

Метод SLR використовується в розділі 3. SLR являє собою перелік поточних досліджень підходи, методи та техніки, що використовуються в еволюції SOA. Результат SLR також визначає поточні дослідницькі проблеми в спадщині для модернізації SOA та надає майбутні напрямки досліджень для вирішення цих проблем дослідження.

У цій дипломній роботі ми також використали вивчення літератури як метод дослідження. Метод вивчення літератури дуже нагадує SLR, оскільки обидва методи дослідження використовуються для отримання певних знань або розуміння певної теми дослідження. Проте дослідження літератури може бути не таким повним і дійсним, як SLR, оскільки дослідження літератури є набагато менш формальним у тому сенсі, що воно дає більше свободи в зборі відповідних досліджень та аналізі їх змісту. Тим не менш, вивчення літератури є ефективним і результативним методом отримати загальне уявлення про тему дослідження. У Розділі 4 ми використовували дослідження літератури для перевірки структурованого процесу, а в Розділі 6 метод дослідження літератури використовується для визначення поточної академічної точки зору застарілих систем і модернізації системи застарілого програмного забезпечення.

Пошук літератури на основі «сніжного кома» — це метод визначення додаткових релевантних статей за допомогою списків посилань набору ідентифікованих статей [29]. [27] використовують сніжну кулю для пошуку

відповідної літератури та пропонують два типи: зворотна сніжна куля – використання списку посилань на статтю для визначення нових статей для включення, і пряма сніжна куля – ідентифікація нових статей на основі тих статей, які цитують статтю, що розглядається. Зворотний підхід «сніжної кулі»

використовується через те, що часу та зусиль, необхідних для проведення пошуку літератури, порівняно менше. Jalali & Wohlin [18] порівнюють SLR із зворотним підходом "сніжної кулі" та стверджують, що підхід зворотної "сніжної кулі" простий у використанні та вимагає менше часу та зусиль.

Метод обґрунтованої теорії

Обґрунтована теорія (Grounded Theory, GT) — це систематичний, індуктивний і порівняльний підхід до ітераційної розробки теорії на основі даних. На відміну від гіпотетико-дедуктивного методу, коли дослідник має заздалегідь визначену гіпотезу на початку дослідження, ГТ є дослідницьким, спрямованим на відкриття нових перспектив і уявлень. Адольф та ін. [3] стверджують, що GT є чудовим методом для вивчення програмної інженерії та створення теорій, які мають відношення до практиків, і стає все більш популярним у дослідженнях програмної інженерії [3]. Наприклад, Coleman et al. [5, 6] використовують GT, щоб зрозуміти вдосконалення процесів програмного забезпечення в ірландських компаніях, що випускають програмне забезпечення; Хода та ін. [23, 22] прийняти GT для вивчення людських аспектів розробки програмного забезпечення; Дедрік та ін. [76] використовувати GT для вивчення впровадження платформ з відкритим кодом у галузі; Анжела та ін. [86] використовувати GT для розуміння практик, орієнтованих на клієнта, у eXtreme Pro gramming (XP); Баласубраманіам та ін. [20] використовувати GT для визначення факторів, які впливають на процеси розробки програмного забезпечення в Інтернеті; Hutchinson та ін. [25] використовують метод GT для документування технічних, організаційних і соціальних факторів, які впливають на реакцію організацій на розробку, керовану моделлю (MDE) у промисловості.

Однією з цілей цієї роботіє виявлення нових перспектив і уявлень про

застарілі системи програмного забезпечення та модернізацію (застарілого) програмного забезпечення з точки зору практиків. Ми не маємо жодної чіткої гіпотези щодо сприйняття практиками застарілих програмних систем та їх модернізації, радше маємо намір створити теорію. Крім того, існує мало емпіричних доказів, задокументованих в академічних колах щодо сприйняття

практиками застарілих програмних систем. Адольф та ін. [3] підкреслюють, що GT є корисним для досліджень у сферах, які раніше не вивчалися, або де нова перспектива може бути корисною. Дотримуючись вказівок щодо інтерв'ю, ми починаємо збирати дані з опитування практиків, які мають досвід роботи із застарілими програмними системами, і транскрибуємо інтерв'ю. Потім транскрипцію аналізують, щоб знайти шаблони, і відповідно призначаються коди для позначення понять, які, у свою чергу, об'єднуються в категорії. Отримані поняття та категорії пов'язані між собою за допомогою пам'яток, що веде до уточнення теорії. Процес триває, доки категорії не будуть «насичені» – нові коди/категорії не додаються навіть із новими інтерв'ю. Згенерована теорія/знахідка потім порівнюється з літературою та/або підтверджується, якщо потрібно, охоплюючи нові перспективи та ідеї, а не підтверджуючи існуючі.

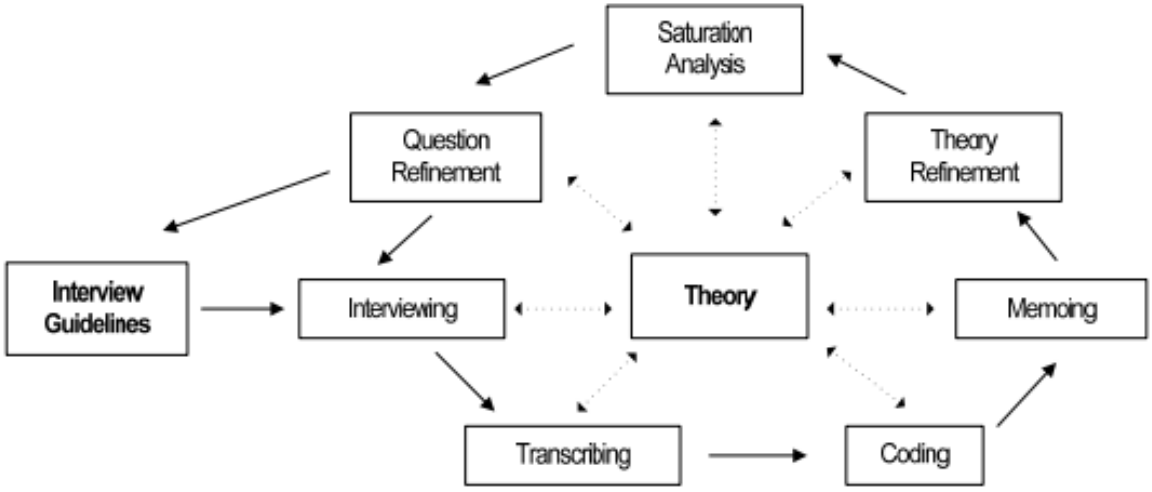

Рисунок 1.7 - Метод обґрунтованої теорії

Опитування — це комплексна система збору інформації для опису, порівняння або пояснення знань, ставлень і поведінки. У цьому дослідженні опитування використовується для тріангуляції даних [23] для підтвердження результатів інтерв'ю методу GT. Процес тріангуляції даних – метод, який використовує більше одного джерела даних або збирає ті самі дані в різних випадках – зазвичай використовується для надання підтверджуючих

(суперечливих) доказів і додатково допомагає покращити валідність результатів емпіричного дослідження. Процес тріангуляції даних все частіше використовується в дослідженнях програмної інженерії. Деякі останні приклади, такі як Greiler et al. [11] і Beller et al. [20] використовують опитування як вторинне джерело даних для покращення перевірки результатів з іншого первинного джерела.

Перевірка

Вирішальним аспектом проведення будь-якого дослідження є сувора операція всього дослідницького процесу та перевірка його, коли це можливо [10]. Перевірка досліджень і даних стає все більшою проблемою в науковому співтоваристві. У статті, опублікованій у журналі Nature у жовтні 2011 року, зазначено, що неправомірна поведінка (тобто фальсифікація або фабрикація, (само)плагіат) становить 44%. Чесна помилка (тобто чесна розбіжність у плані, виконанні, інтерпретації чи оцінці методів або результатів дослідження) становить 28% від загальної кількості відкликань у науковому співтоваристві [27]. Методи дослідження, які використовуються в цій дисертації, є переважно емпіричними, і тому вимагають суворої перевірки та оцінки. Наприклад, Yin [27] і Eisenhardt [8] стверджують, що валідація, зокрема валідація методу дослідження та валідація даних, є важливим аспектом методу дослідження тематичних досліджень. У цій роботіми приділили особливу увагу операційалізації вибраних методів дослідження, перевірці даних і зменшенню упередженості дослідження протягом усього дослідження.

У разі застосування вибраних методів дослідження ми ретельно дотримувалися доступних рекомендацій щодо дослідження. Наприклад, у тематичних дослідженнях ми дотримувалися вказівок Runeson & Host [23],

створюючи протоколи тематичних досліджень і чітко документуючи джерела даних, якщо це можливо. У методі дослідження, заснованому на SLR, рекомендації щодо проведення SLR Kitchenham et al. [13] суворо дотримувалися. У випадку інтерв'ю учасникам надається протокол інтерв'ю, щоб підготувати загальне розуміння того, що буде обговорюватися. Доведено, що такі протоколи

(протокол дослідження та протокол інтерв'ю) є надзвичайно важливими, оскільки вони забезпечують спільну основу для розуміння використовуваних термінів і методів (наприклад, концепція застарілих систем розуміла по-різному для різних підприємств). Коли це можливо, розроблені артефакти перевіряються експертами та згодом вдосконалюються.

У разі перевірки даних ми зосередилися на документуванні джерел даних, коли це можливо. Наприклад, більшість інтерв'ю, проведених у контексті цього дослідження, було записано та транскрибовано з дозволу опитуваного. У разі виникнення непорозумінь і для подальшого підтвердження опитуваних консультувалися електронною поштою, щоб роз'яснити питання – метод, відомий як кооперативне опитування [22]. Артефакти, зібрані під час збору даних, такі як форми, форми класифікації SLR, суперечки між дослідниками під час класифікації первинних досліджень, документи з тематичних досліджень документуються. Щоб пом'якшити упередженість дослідження на етапі аналізу дослідження, у всій роботіобговорюються різні типи валідності. [19] описують валідацію як захід для забезпечення обґрунтованості вимірювань з реальним світом. Відповідно до [27], ідентифіковано наступні чотири різні типи дійсності:

1. Конструктивна валідність: відображає, якою мірою досліджувані оперативні показники справді представляють те, що було призначено для вимірювання.

2. Внутрішня валідність: відображає ступінь виправданості причинно-наслідкового висновку на основі дослідження.

3. Зовнішня валідність: стосується можливості узагальнення результатів і вимірює ступінь можливості узагальнити результати.

4. Надійність: питання про те, наскільки дані та аналіз залежать від конкретних дослідників.

Крім того, ми також використали тріангуляцію даних для підтвердження результатів деяких досліджень.

1.4 Висновки по розділу

У цьому розділі було розглянуто основи еволюції програмного забезпечення, характеристики застарілих систем та ключові стратегії й методи їх модернізації. Особлива увага приділена сервісно-орієнтованій архітектурі як перспективній цільовій платформі, яка дозволяє ефективно повторно використовувати існуючі активи та зменшувати витрати на обслуговування.

РОЗДІЛ 2. ІНЖЕНЕРНІ МЕТОДИ МОДЕРНІЗАЦІЇ ТА ПОВТОРНОГО ВИКОРИСТАННЯ ЗАСТАРІЛИХ ПРОГРАМНИХ КОМПОНЕНТІВ

2.1 Огляд підходів до модернізації

Велика кількість підприємств залежить від критично важливих для бізнесу систем для консолідації бізнес-інформації, які були розроблені протягом останніх трьох десятиліть або більше з використанням мов програмування 3GL, таких як COBOL, RPG, C, C++ [22]. Ці системи називаються застарілі системи. За оцінками, більше 80% світового бізнесу працює на COBOL, і 50-70% загальних витрат на IT витрачається на підтримку цих систем [16]. Успадковані системи зараз є перепоною на шляху розвитку IT-інфраструктури на підприємстві через їхні добре відомі недоліки, такі як негнучкість і складність обслуговування [32].

Однак підприємства все ще покладаються на ці успадковані системи, оскільки вони зазвичай впроваджують складні основні бізнес-процеси та високий ризик, пов’язаний з необхідними змінами [13]. Оскільки застарілі системи є життєво важливими для продовження бізнесу на підприємствах, зростає імпульс для розвитку цих систем у нових технологічних середовищах [32]. Нещодавно сервіс-орієнтована архітектура (SOA) з’явилася як багатообіцяючий архітектурний стиль, який дозволяє існуючим застарілим системам демонструвати свою функціональність як послуги, не вносячи значних змін у самі застарілі системи [17]. Перехід із застарілих систем на SOA може бути вигідним з економічної та технічної точки зору. З економічної точки зору підприємства постійно стикаються з прискоренням темпів змін, таких як внутрішньоорганізаційні зміни, зміни ринкових вимог і можливостей і, як наслідок, зміни у співпраці підприємств. Перехід із застарілої версії на SOA дає змогу застарілим системам адаптуватися до таких змін [15] і має на меті зниження витрат на технічне обслуговування [20]. З технічної точки зору, заявлено безперебійну співпрацю підприємства за допомогою композиції

сервісів та гетерогенної інтеграції додатків усередині та за межами підприємств [148].

У літературі повідомлялося про кілька підходів до міграції застарілих систем на SOA та технологію веб-служб. Значна кількість таких підходів зосереджена на розробці допоміжної технології для вирішення проблеми технічної міграції (тобто методи реалізації, щоб показати застарілий код як послугу) [18, 6, 8, 15, 7]. Інші підходи зосереджені на розробці стратегії міграції для визначення доцільності міграції. Така здійсненність визначається на основі характеристик існуючих успадкованих систем щодо їхнього потенціалу для показу як послуг та вимог цільової системи SOA [27]. Однак метод міграції із застарілою версією вимагає консолідації обох вищезазначених точок зору (тобто можливості міграції та підтримуючої технології) [7], яка, як нам відомо, досі відсутня.

У цій главі метод міграції із застарілої версії на SOA, який далі називається Метод ServiciFi, розроблено так, що поєднує в собі можливість переходу на SOA та розробку технології підтримки спадщини. Метод serviciFi розроблений шляхом складання фрагментів існуючих сервіс-орієнтованих методів розробки за допомогою інженерії методів [3]. Для розробки допоміжної технології використовується концепція нарізки [10], щоб полегшити вилучення послуг із застарілого коду.

2.2 Технічні передумови та виклики

Метод serviciFi розроблено відповідно до наукового проектування в дослідженні інформаційних систем, запропонованого [11]. Метод serviciFi спочатку розробляється та оцінюється за допомогою огляду експертів, а потім двох прикладів. Інженерія методів використовується для розробки методу serviciFi шляхом збирання дій існуючих методів сервіс-орієнтованої розробки. У наступних підрозділах детально описано підхід до розробки методу та пов’язані

з ним концепції, які використовуються під час розробки методу servicFi. Крім того, нарізка концепції також пояснюється як допоміжна технологія для міграції.

Інженерія методів для розробки інформаційних систем - це підхід до побудови передових методів розробки шляхом повторного використання частин існуючих методів [9]. У роботі Брінккемпера [9] а «метод» визначається як підхід до виконання методу розробки системи, що складається з вказівок і правил, систематично структурованих у діяльності з розробки з відповідними продуктами. Діяльність розробки та відповідні продукти називаються «фрагменти методу». Метод інженерного підходу використовується для розробки методу serviciFi завдяки тому факту, що повторне використання існуючих і перевірених фрагментів методу з існуючих методів сервіс-орієнтованої розробки економить час і зменшує проблему адаптації (тобто легко адаптувати до існуючих стандартів/методів). Особливою стратегією розробки методів є ситуаційна інженерія методів на основі складання [4], яка включає етап створення бази методів у випадку, якщо вона не існує. База методів - це репозиторій, де можна зберігати та отримувати фрагменти методів [9]. Щоб створити метод serviciFi, немає існуючої бази методів, з якої фрагменти методів можна використовувати повторно. Отже, розробка ситуаційного методу на основі складання використовується для створення бази методу для методу serviciFi. Розробка ситуаційного методу на основі складання включає наступні кроки, які виконуються для створення методу serviciFi.

Потреба в методі serviciFi описана в розділі 2.1. Метод serviciFi має поєднувати можливість міграції та розробку допоміжної технології для вилучення застарілого коду та надання їм послуг. Крім того, витягнуті служби повинні відповідати поточним стандартам SOA, щоб полегшити інтеграцію гетерогенних програм на підприємствах/всередині них.

Вибір методів-кандидатів

На цьому кроці вибираються сервіс-орієнтовані методи розробки, що містять відповідні фрагменти методів. Виходячи з більшої кількості цитувань (популярності), наявності документації та повноти методу, вибрано наступні

методи сервіс-орієнтованої розробки: Service-Oriented Design and Development Methodology (SODDM) [208], Web Service Implementation Methodology (WSIM) [16], Service-Oriented Modeling and Architecture (SOMA) [12]. Далі ці методи називаються «кандидатські методи».

Аналізуйте методи-кандидати та зберігайте відповідні фрагменти методів у базі методів

На цьому кроці база методів заповнюється відповідними фрагментами методів, отриманими з трьох методів-кандидатів. Для кожного кандидатського методу створюється діаграма результатів процесу (PDD) шляхом застосування техніки метамоделювання, як описано Weerd та ін. [27]. ПТД кандидатських методів зображує кожну діяльність і результати кожного етапу. Деталі техніки метамоделювання для розробки PDD описані в Reijnders et al. [22]. Щоб скласти враження про те, як представлено ПТД, ПТД методу serviciFi (див. рис. 2.4) представлено як приклад. PDD складається з двох інтегрованих діаграм: подання процесу в лівій частині діаграми базується на діаграмі активності UML, а представлення результатів у правій частині діаграми базується на діаграмі класів UML. Діаграма перегляду процесу/результату складається з різних типів діяльності/концепцій. Ці дії/концепції зображені на малюнку 2.1 і пояснені наступним чином:

- Стандартна діяльність/концепція: Стандартна діяльність/концепція не містить інших дій/концепцій.
- Відкрита діяльність/концепція: Відкрита діяльність/концепція містить додаткові дії/концепції.
- Закрита діяльність/концепція: Діяльність/концепція, дії/концепції якої не розроблені, оскільки вони невідомі або не мають відношення до конкретного контексту.

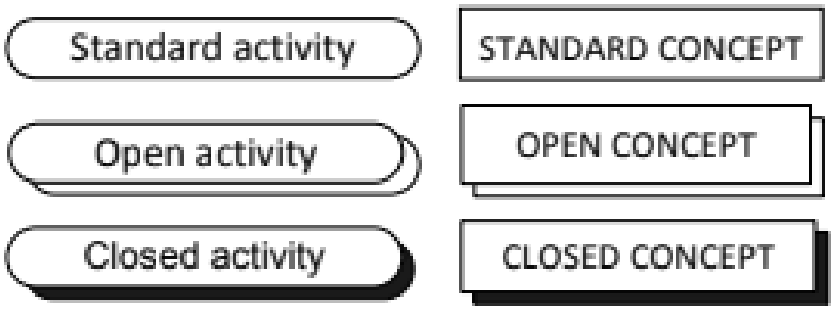

Рисунок 2.1 - Типи діяльності та концепції

ПТД методів-кандидатів описує кожну діяльність і результати кожної фази, які зберігаються в базі методів як фрагменти методу. На цьому кроці корисні фрагменти методу з бази методу вибираються та збираються для формування методу serviciFi. Цей крок розділений на три піддіяльності.

Таблиця 2.1

Порівняння фаз

#	WSIM	SODDM	SOMA
1	Requirements	Planning	Business Modeling & Transformation
			Solution Management
2	Analysis	Analysis & Design	Identification
3	Design		Specifications
4	Coding	Construction & Testing	Realization
	Testing		Implementation
5	Deployment	Provisioning	Deployment, Monitoring & Management
		Deployment	
		Execution & Monitoring	

По-перше, необхідно визначити загальні фази методу serviciFi, що робиться шляхом порівняння та аналізу фаз трьох методів-кандидатів. Порівняння фаз зображено в таблиці 2.1. У результаті цього порівняння виявлено п'ять фаз методу serviciFi, що нагадують фази.

По-друге, порівняння методів між фрагментами методів, що зберігаються в базі методів, виконується за допомогою матриці порівняння методів [27] для створення так званого “супер-метод". Суперметод містить дії, які вважаються повторно використаними для методу serviciFi. Створення

суперметоду передбачає покрокове порівняння всіх дій і результатів між трьома методами-кандидатами в базі методів. Кожну дію на тій самій фазі оцінювали на основі їх опису, щоб визначити, чи була ця діяльність:

- Виходить за межі області, так що діяльність відкидається.
- Дорівнює іншій діяльності, так що лише одна з дій включена в суперметод.
- Повністю міститься в іншій діяльності, щоб прийняти рішення щодо обсягу.
- Нерелевантно на поточній фазі, тому діяльність відкидається.
-

Таблиця 2.2

Витяг із фази ініціації проекту матриці порівняння методів

WSIM	SODDM	SOMA
Determining the need of web service	**Analyzing the business needs**	-
-	**Review current technological landscape**	-
Elicit web service requirement Manage Web service requirements Model usage scenario	**Conceptualize the new requirements**	-
-	Manage project deliverables and resources	**Initiate project management**
*****Prepare test cases for user acceptance test and system test*****	-	-
-	-	*Define business architecture and models*
-	-	*Select solution templates and patterns*
-	-	*Conduct method adoption workshop*

У таблиці 2.2 наведено уривок матриці порівняння методів фази ініціації проекту. Дії, виділені жирним шрифтом, являють собою зібрані фрагменти методу для формування суперметоду. Діяльність всередині курсив виходять за рамки. Діяльність у межах * * не актуальна на поточному етапі. Діяльності, схожі або якщо їх об’єднати, подібні до вищої деталізації, представлені в одному рядку матриці порівняння методів. Відповідний фрагмент суперметоду, що представляє фрагмент матриці порівняння методу (табл. 2.2), зображено на рисунку 2.2. серед «Аналіз бізнес-потреб» СОДДМ і «Визначити потребу у веб-

службі» WSIM, які є функціонально схожими видами діяльності, вибирається більш ранній, оскільки найменування є більш значущим. The «Концептуалізація нових вимог» активність SODDM вибрано, оскільки вона представляє більш детальну активність порівняно з трьома подібними видами діяльності WSIM (тобто, «Вимоги до веб-служб Elicit», «Керування вимогами до веб-сервісу», «Сценарій використання моделі»). Нарешті, «Ініціювати управління проектом» активність SOMA вибирається в порівнянні з «Управління результатами та ресурсами проекту» діяльності СОДДМ як найменування в діяльність має більший зміст. Шляхом порівняння діяльності та результатів кожної фази створюється суперметод методу serviciFi. По-третє, суперметод тепер складається з фрагментів методу зібрані з трьох методів-кандидатів. Але, щоб адаптувати супер-метод для міграції, додаються деякі інші дії та результати, такі як аналіз витрат і вигод [246], ідентифікація служб сторонніх розробників, які мають схожі функціональні можливості з послугами, які мають бути виділені, а також методи пріоритету для визначення пріоритету вилучення ідентифікованих послуг. Крім того, зібрані фрагменти методу перейменовано для адаптації до контексту міграції. Наприклад, «Аналізуйте потреби бізнесу» суперметоду (див. рис. 2.2) перейменовано на «Визначте цілі проекту», «Огляд технологічних ландшафтів» до «Аналіз технологічного ландшафту», і так далі. На основі цих змін у супер-методі завершується метод serviciFi, який пояснюється в розділі 2.3.

Рисунок 2.2 і таблиця 2.2 є лише фрагментами суперметоду та матриці порівняння методів відповідно. Деталі суперметоду та матриці порівняння методів не включені в цей розділ через причину стислості. Їх подробиці були описані в роботі Reijnders et al. [22].

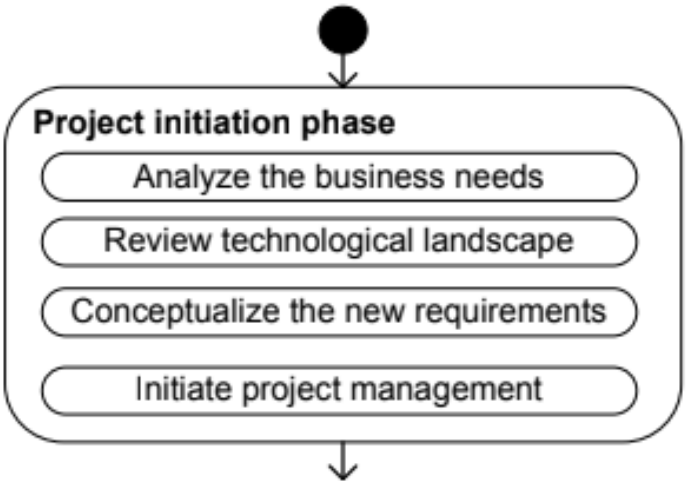

Рисунок 2.2 - Фрагмент етапу ініціації проекту суперметоду

Метод serviciFi спрямований на повторне використання існуючого застарілого коду для отримання послуг. The «Побудова та тестування сервісу» етап методу serviciFi відрізняється відповідним "реалізувати це" етап трьох методів-кандидатів. The «Побудова та тестування сервісу» фаза повинна включати заходи для сприяння видобутку застарілого вихідного коду, тоді як відповідні «Реалізація» фаза трьох методів-кандидатів спрямована на створення нових послуг з нуля. Нарізка концепції використовується як техніка реалізації (тобто допоміжна технологія) для вилучення застарілих кодів і надання їх як послуг у «Побудова та тестування сервісу» фаза.

Концепція нарізки [10] поєднує дві методики з (ре)інжинірингу програмного забезпечення та сфери обслуговування: нарізка програми і призначення концепції створити "Концептуальний фрагмент виконуваного файлу"(ECS). Розрізання програм [29] — це добре відома техніка аналізу коду, яка використовується для визначення та абстрагування найменшої можливої підмножини програми, яка може виконувати очікувану функціональність. Призначення концепції [30] — це техніка, яка призначає окремі орієнтовані на людину концепції частинам вихідного коду. Обидва методи використовувалися як методи вилучення вихідного коду, які приймають критерій і вихідний код програми як вхідні дані та видають частини вихідного коду. програма як результат, однак, критерій вилучення програмного нарізання виражено на дуже

низькому рівні для побудови критерію нарізання, наприклад, використання програмних змінних, що робить нарізку складно застосовувати. У призначенні концепції критерій вилучення виражається на рівні домену, що робить його більш практичним у застосуванні, але, на відміну від нарізки програми, витягнутий код не можна виконати як окрему (під)програму. Щоб досягти сукупних переваг, одночасно подолавши окремі недоліки, [10] поєднав ці дві методики, як нарізку концепції, і успішно витягнув виконуваний вихідний код із застарілого коду.

2.3 Методологія ServiciFi для реінжинірингу

Метод serviciFi зображено на рисунку 2.4 як PDD. У наступних підрозділах детально описано кожну фазу та її складові дії. The «Ініціація проекту» Фаза виконує оцінку життєздатності міграції спадщини на SOA шляхом аналізу технічної та економічної здійсненності. Фаза починається з "Визначте цілі проекту"діяльність, яка визначає, які функціональні можливості застарілої системи мають бути представлені як послуги. Друга діяльність, "Проаналізуйте технологічний ландшафт», аналізує технічні аспекти існуючих застарілих систем, такі як мова програмування, яка використовується для створення застарілої системи, доступність документації чи ресурсів, а також вимоги до цільової системи SOA. «Аналіз технологічного ландшафту» діяльність, в «аналіз портфоліо» Виконується [246] визначених функціональних можливостей, які мають бути представлені як послуги. Аналіз портфоліо оцінює як технічну інформацію, так і бізнес-цінність визначених функцій. Технічна інформація включає загальне функціонування застарілої системи та функціональні можливості, наявні в застарілій системі, з якої ідентифіковані функціональні можливості потрібно перенести. Бізнес-цінність визначених функцій включає попередні переваги, які можна отримати, показуючи ідентифіковані функції як послуги. The «Аналіз портфоліо» дає загальний огляд застарілої системи, її функціональних можливостей та економічних переваг.

Аналіз портфоліо виконується шляхом опитування розробників, якщо такі є, та/або перегляду доступної документації та/або поточних користувачів застарілих систем. Бізнес-вартість вказує на життєздатність бізнес-інвестицій і попередню віддачу від інвестицій, яка розраховується як вартість обслуговування (якщо можливо). Бізнес-цінність міграції аналізується шляхом опитування бізнес-менеджерів і дослідження потреб ринку. Цілі проекту та аналіз технологічного ландшафту передбачають нові вимоги до проекту, які задокументовані як вимоги в «Викликати нові вимогиНарешті, «План управління проектом» зазначено. Результатами фази ініціації проекту є план управління проектом і (не)схвалення проекту міграції.

Ця фаза зосереджена на визначенні потенційних послуг, які відповідають вимогам, описаним у розділі «ініціювання проекту"фаза. Перша діяльність, "Проаналізуйте ситуацію як є", є відкритою діяльністю, що складається з трьох підактивностей, як показано на рисунку 2.3.

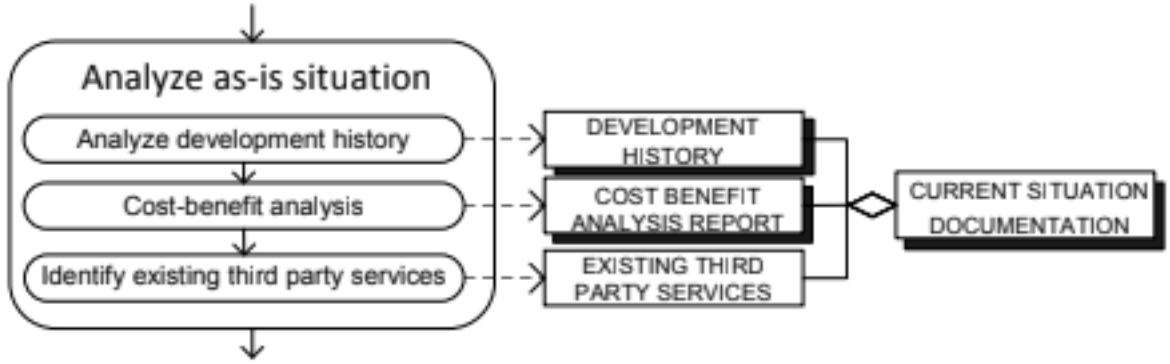

Рисунок 2.3 - Аналіз ситуації «як є».

"Проаналізуйте історію розвиткуПіддіяльність досліджує артефакти застарілої системи, такі як документи вимог, діаграми UML, діаграми даних, вихідний код, діаграми класів, графіки системних залежностей і навіть коментарі в коді в деталях порівняно з «Проаналізуйте технічний ландшафт" діяльність "Ініціація проектуУся ця інформація дає змогу краще зрозуміти історію розробки, а також функціональні можливості, що містяться в застарілій системі. The «Аналіз витрат і вигод», як запропонував Снід [26], виконується

для оцінки вартості міграції. Аналіз витрат і вигод виконується для порівняння витрат на міграцію з очікуваними вигодами. Як правило, на цьому етапі порівнюють переваги міграції, реконструкції та бездіяльності. The «Аналіз витрат і вигод» піддіяльність визначає, чи проект міграції є економічно життєздатним. "Визначте існуючі сторонні служби"Діяльність досліджує, чи служби, які метод міграції має на меті отримати, уже доступні на ринку. Наявність таких існуючих послуг може або надати можливості для створення складних функціональних можливостей, або вже прийняти рішення про повторне використання цих існуючих послуг, а не вилучення із застарілої системи, якщо це можливо. Наприклад, якщо одна з функціональних можливостей, яку потрібно витягти як послугу, є "Перевірка кредитної картки", а потім повторно використовувати доступні безкоштовні веб-сервіси, наприклад ValidateCreditNumber 15 може бути економним.

Одного разу "Проаналізуйте ситуацію як є» задокументовано, наступною діяльністю є «Визначте кандидатів на послугиНаразі ця діяльність виконується вручну на основі функцій, визначених у застарілому коді, відповідно до вимог, визначених у «ініціювання проекту". Цілком можливо, що можуть існувати невідповідності між визначеними функціональними можливостями в застарілому коді та вимогами до послуг, які будуть виставлені. Такі невідповідності задокументовано в "Порівняння цілей", за допомогою якого можна визначити, чи може визначена служба-кандидат відповідати бізнес-вимогам. Залежно від порівняння цілей, проект може бути скасовано, якщо ідентифіковані послуги-кандидати не відповідають вимогам. Для кожної з ідентифікованих служб-кандидатів необхідно визначити деталізацію, яка виконується в "Визначте деталізацію послуги". Деталізація служби визначає, чи потрібно ідентифіковані служби-кандидати видобувати як атомарну службу чи як складену службу, що представляє склад атомарних служб. Обидві деталізації мають власні переваги, такі як вилучення атомарних служб дозволяє створювати нові функціональні можливості в майбутньому і, отже, збільшує можливість повторного використання витягнутих служб. У той час як вилучення складених

служб дозволяє підтримувати кілька служб після розгортання в інфраструктурі обслуговування, а отже, зменшує витрати на технічне обслуговування.Встановити пріоритетність послуг", у якому визначені послуги-кандидати встановлюються за пріоритетом на основі вимог і потреб бізнесу. Техніка пріоритету MoSCoW [14] використовується для визначення та створення списку пріоритетів для ідентифікованих служб-кандидатів. На основі списку пріоритетів плануються ітерації розробки. Перша ітерація починається з функціональністю з найвищим пріоритетом. Після кожної ітерації список пріоритетів переоцінюється. Ітерація сприяє поступовому розвитку міграції.

«Визначте існуючі сторонні служби» піддіяльність в «Аналізуйте ситуацію як є» діяльність. Такі служби сторонніх розробників, які беруть участь у поточній ітерації, потрібно зіставляти з існуючими типами даних і іменами змінних застарілого коду. Ця операція надає огляд вхідних і вихідних повідомлень, необхідних для цих служб-кандидатів. Після завершення зіставлення між існуючими службами та послугами-кандидатами послуги-кандидати деталізуються на технічному рівні. Базуючись на роботі Джона Стона [15], для завершення специфікації послуги на технічному рівні виконуються наступні три кроки: (i) Конструктивна специфікація який визначає необхідні операції, які представляють функціональність служб-кандидатів, і повідомлення, які передаються через операції, дотримуючись застарілого коду, (ii) Поведінкова специфікація що визначає сервісні інтерфейси, які використовуватиме клієнт разом із необхідними вхідними, вихідними повідомленнями та операціями, і (iii) Специфікація політики що позначає твердження політики та обмеження для кожної конкретної служби.

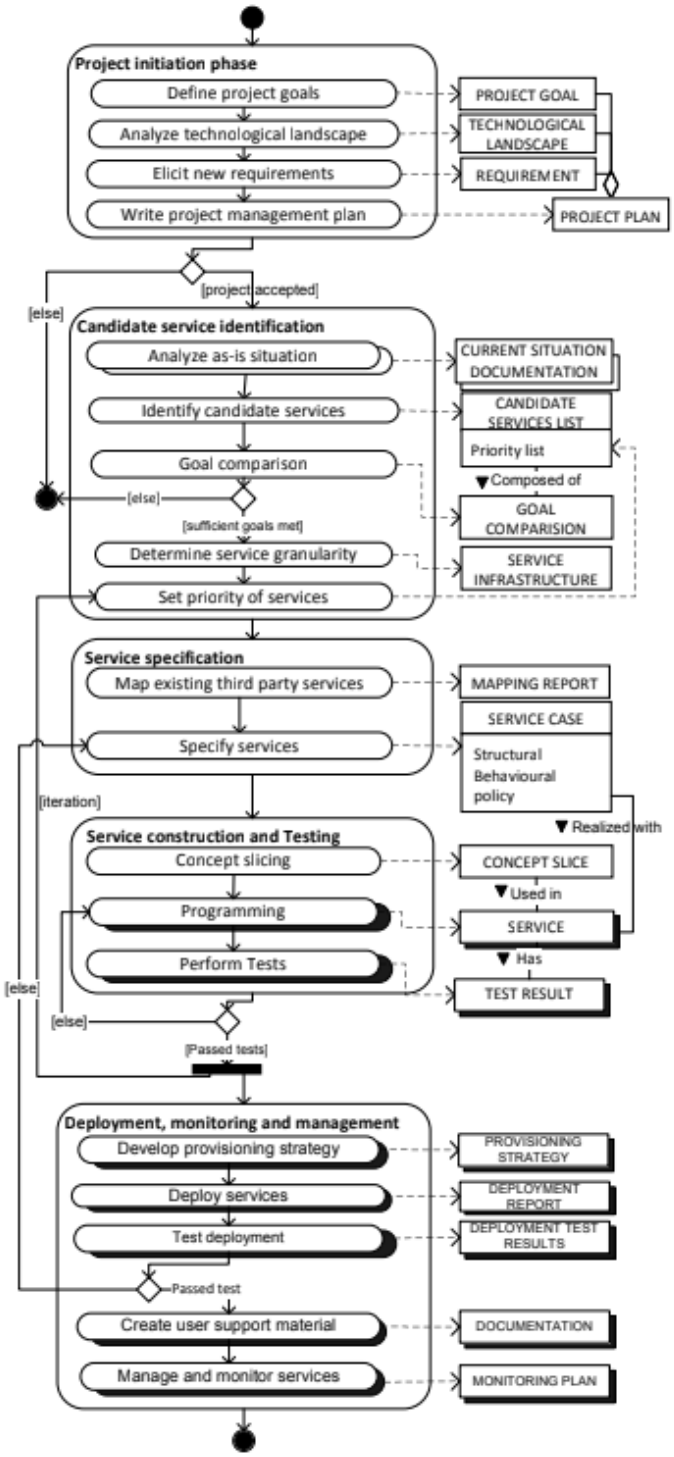

Рисунок 2.4 - ПТД методу serviciFi

"Сервісне будівництво та тестування"Фаза стосується вилучення вихідного коду із застарілої системи.Нарізка концепції"Техніка використовується для вилучення застарілого коду як виконуваного концептуального фрагмента (ECS). Ця техніка успішно використовується

Harman та ін. [14] для вилучення виконуваних (під)програм із фінансової програми на основі COBOL, а [18] з бухгалтерського програмного забезпечення на основі C. За діяльністю зрізання концепції слідує "Програмування", щоб полегшити виконання ECS, якщо потрібно. Наприклад, ECS може знадобитися розгортати в інших середовищах, тому ECS може знадобитися рефакторинг та/або системні виклики, наявні в ECS, можливо, доведеться замінити або перепрограмувати. Такі зміни виконуються в «Програмування» діяльність. Нарешті, кожен ECS тестується, щоб визначити, чи він функціонує відповідно до застарілого коду. в "Виконайте тести", виконуються такі тести: модульні тести, функціональні тести та системні тести залежно від того, як розроблено ECS. У разі автоматичного вилучення ECS за допомогою інструментів нарізки концепції виконуються лише функціональні тести. У випадку, якщо ECS розроблено за допомогою ручного вилучення, тоді потрібно виконати всі вказані тести. Коли ECS проходить ці тести, можна з упевненістю припустити, що витягнутий код справді задовольняє необхідні функції та готовий бути розгорнутим як сервіс в інфраструктурі сервісу. У той же час ініціюється ітерація для наступного сервісу-кандидата в списку пріоритетів.

"Розгортання, моніторинг і управління" етап стосується розгортання ECS як служби. Початкова діяльність "Розробіть стратегію забезпечення"полегшує використання послуг як з технічних, так і з бізнес-аспектів споживачем [21]. Надання послуг зазвичай включає такі дії, як публікація послуг у каталозі, створення версій послуг, а також вимірювання та виставлення рахунків за використання послуг [14]. Потім служби розгортаються в інфраструктурі послуг і тестуються. На основі результатів "Тестове розгортання", або вилучення служби має бути виконано знову, якщо зазначені функції не виконуються, або служба готова до використання. Згодом створюються матеріали підтримки користувачів, такі як документація. Крім того, щоб забезпечити належне функціонування розгорнутих служб, підтримка керування та моніторингу надається в "Управління та моніторинг послуг” діяльність.

2.4 Оцінка ефективності методів модернізації

Для того, щоб оцінити метод serviciFi, спочатку було проведено вісім напівструктурованих інтерв’ю з експертами. Пізніше метод serviciFi було застосовано до двох прикладів, щоб оцінити його на практиці. У наступних підрозділах детально описано кожен метод оцінювання.

Експертний огляд

Метод оцінюється кількома експертами з промисловості та наукових кіл шляхом проведення напівструктурованих інтерв’ю. Цей метод був обраний тому, що він включає суміш відкритих і конкретних запитань, призначених для отримання не тільки передбаченої інформації, але й несподіваних типів інформації [242]. Таблиця 2.3 описує деталі експертів. Імена зберігаються анонімними, а досвід (у роках) у відповідній галузі чітко запитували перед проведенням інтерв’ю. Різниця в досвіді експертів дозволила оцінити метод serviciFi з різних точок зору практиків програмного забезпечення. Перед співбесідою експертам було надано PDD методу serviciFi та відповідну документацію. Метод спочатку пояснювався кожному експерту перед проведенням фактичного інтерв'ю. Пізніше експертів попросили дати відгук або зауваження щодо методу. Інтерв’ю проводилися англійською мовою і кожне з них займало від 80 до 120 хвилин. Кожне інтерв’ю записувалося, а потім транскрибувалося. Інтерв'ю було проведено для оцінки запропонованого методу на основі п'яти показників якості, запропонованих Brinkkemper et al. [40] під час проектування методів за допомогою інженерії методів. П'ять показників якості описуються наступним чином:

- Повнота: Метод serviciFi — це збірка фрагментів методу існуючих сервіс-орієнтованих методів розробки, тому експертів запитали, чи метод serviciFi фіксує повні дії/фази сценарію міграції.
- Застосовність: Експертів запитали, чи суперечать будь-які дії/фази в методі serviciFi один одному таким чином, що перешкоджає застосуванню в будь-якому проекті міграції.

- Ефективність: Експертів запитали, чи метод serviciFi є ефективним і чи не містить жодного повторення фаз/дій, що збільшило б витрати та зусилля.
- Консистенція: Експертів запитали, чи були фази/діяльності узгодженими (тобто семантично правильними та значущими) одна з одною.
- Надійність: Експертів запитали, чи надійний метод serviciFi для застосування в будь-якому реальному успадкованому проекті міграції SOA.

Таблиця 2.3

Інформація про експертів

Name	Expertise	Experience	Sector
A	Software product manager	5	Industry
B	Application manager	5	Industry
C	Software engineering researcher	5	Academia
D	Migration from legacy to SOA/cloud	5	Industry
E	Migration from legacy to SOA	3	Industry
F	Software migration	3	Industry
G	SOA researcher	7	Academia
H	Requirement engineering researcher	6	Academia

Після того, як усі експерти були опитані, для аналізу результатів було використано постійний порівняльний аналіз (CCA) [26, 3]. Метод CCA використовується для створення знань із джерела даних шляхом уникнення суб'єктивної інтерпретації [26] (тобто інтерпретації даних відповідно до цілей дослідження). Результат аналізу було розділено на дві категорії: (i) запити на незначні зміни які включали такі зміни, як перейменування назв діяльності, перейменування результатів та (ii) запити на серйозні зміни які включали такі зміни, як додавання/видалення дій, додавання ітерацій серед фаз, додавання/видалення розгалужень. ПТД, зображена на рисунку 2.4, уже містить коригування, внесені після CCA.

У відповідь на повнота і послідовність показників якості, 75% (6 із 8 експертів) погодилися, що фази методу serviciFi повні та послідовні. Однак додавання «Аналіз витрат і вигод» піддіяльність в "Ситуація як є" діяльність «Ідентифікація кандидата на службу» була підкреслена фаза. Крім того, в результаті аналізу було зроблено акцент на використання методу ітераційної розробки. Один із експертів описав необхідність ітераційного методу розробки

як «процес міграції займає багато часу, тому потрібні ітераційні цикли розробки для включення змін, які відбулися протягом життєвого циклу проекту.» У відповідь на застосовність вимірювання якості, аналіз показав, що майже всі експерти погоджуються щодо застосовності методу, оскільки метод serviciFi містить усі типові фази та дії, необхідні для проекту міграції.

У відповідь на ефективність і надійність показників якості, половина експертів (4 з 8 експертів) не були впевнені в ефективності та надійності методу. Аналіз показав, що на ефективність і надійність можуть впливати інструменти та методи, які використовуються в цьому методі, а також різні фактори застарілих систем, такі як доступність документації, підтримка поточних користувачів для розуміння системи, якість вихідного коду, надмірність у вихідному коді, а також зрозумілість і зручність обслуговування коду. Один з експертів пояснив цю ситуацію так:визначення ефективності залежить від контексту, оскільки фаза ідентифікації служби-кандидата та фаза побудови служби сильно залежать від статусу застарілої системи, наприклад, наскільки добре написаний і підтримується застарілий код.»

Аналіз також підкреслив необхідність міграції для поточних користувачів. Міграція таких користувачів дійсно є важливим аспектом, який не можна недооцінювати [18, 20]. Рекомендується здійснювати належне планування міграції користувачів шляхом проведення навчальних програм [23], однак міграція користувачів виходить за рамки цього дослідження. Загалом аналіз експертних оцінок показав, що більшість експертів погодилися повнота, послідовність, і застосовність заходи якості, але не були впевнені в ефективність і надійність. Для того, щоб оцінити ефективність і надійність вимірювання якості, було проведено два приклади, у яких метод serviciFi використовувався для вилучення послуг із застарілої системи. Подробиці цих прикладів описано в 2.4.3 і 2.4.4.

Початкове тематичне дослідження було проведено в голландській продуктовій компанії, яка використовує застарілу систему на основі COBOL для управління заробітною платою та персоналом. З міркувань конфіденційності

така інформація, як назва компанії, назва продукту та доступні функції застарілих систем залишаються анонімними. Компанія хотіла повторно використовувати одну з функцій, наявних у застарілій системі, як веб-сервіс. Для міграції функціональності використовувався метод serviciFi. Мета проекту полягала в тому, щоб отримати певну функціональність і представити її як веб-сервіс. Аналіз портфоліо «Аналіз технологічного ландшафту» показало, що система була розроблена близько 10 років тому на COBOL. Різні інші функції всієї застарілої системи також залежали від цієї конкретної функції. Метою цієї міграції було залучити більше клієнтів до використання їх програмного забезпечення як послуги. Крім того, компанія прагнула знизити витрати на технічне обслуговування, представивши функціональність як послугу. The «Виявлення нових вимог» діяльність в «Ініціація проекту» Фаза наголошувала на використанні стандартів веб-сервісів, яких слід дотримуватися під час міграції. План проекту був задокументований і «Ідентифікація кандидата на службу» фаза була розпочата. в «Ідентифікація кандидата на службу» фаза, в “Ситуація як є” піддіяльність вказала, що застарілий код складається з 1588 рядків коду з великою кількістю винятків. Наявність оригінального програматора вихідного коду допомогла нам зрозуміти конкретний функціонал. The «Аналіз витрат і вигод» було зроблено компанією, яка показала, що міграція є економічно вигідною та вигідною. The «Ідентифікація існуючих сторонніх служб» не було актуальним у цьому випадку, оскільки компанія хотіла мати власний сервіс, а також «Порівняння цілей» діяльність не була актуальною. Сервіс мав бути витягнутий як атомарний сервіс, щоб його можна було повторно використовувати в майбутньому для будь-якої іншої композиції сервісу. Через виділення однієї послуги список пріоритетів не створено. в «Специфікація послуги» фаза, початкова діяльність «Відобразити наявні сторонні сервіси» не було актуальним для поточної функціональності. Структура, поведінка та політика служби були визначені шляхом консультації з початковим програмістом і продукт-менеджером системи.

«Нарізка концепції» було виконано вручну через відсутність інструменту нарізки для програм COBOL. Використовуючи призначення концепції, було ідентифіковано 14 зі 132 змінних, пов’язаних із концепцією (тобто функціональністю, яку потрібно витягти). На основі цих 14 змінних було застосовано нарізку програми, за допомогою якої ми витягли 426 рядків коду з 1588 рядків коду. Витягнутий код було успішно скомпільовано, і загалом було запущено 240 тестових випадків для перевірки функціональності видобутого коду. Усі результати тестів порівнювали з результатами тестів, запущених на оригінальному коді, і було отримано 100% успішний результат.

Практичний приклад C++

У другому прикладі — програма на C++ під назвою SrnaCalc16 було використано для демонстрації та оцінки запропонованого підходу. Це калькулятор із відкритим вихідним кодом, виданий згідно з GNU General Public License ver.3.0 (GPLv3). SrnaCalc — це простий калькулятор командного рядка з основними математичними функціями, пам’яттю та можливістю створення сценаріїв. The «Аналіз технологічного ландшафту» активність показала наявність таких функцій: оператори для відображення використаних операторів, eval оцінити вираз, getPrecision і setPrecision керувати точністю, пам'ять щоб переглянути вміст пам'яті, додати,встановити, isset, отримати, видалити і читати додавати, змінювати, знаходити, додавати, видаляти та читати змінну пам’яті відповідно.

Метою було видобути eval функціональність, яка обчислює вирази, і надавати її як послугу. The «Викликати нову вимогу» і «Напишіть план управління проектом» діяльність не була актуальною в цьому прикладі. The «Аналіз ситуації як є» діяльність призвела до показників програми, як показано у таблиці 2.4 та графік програмних залежностей, як показано нарисунку 2.5. Граф програмних залежностей був створений за допомогою інструменту Understand [24] і вручну відредагований для візуального збагачення шляхом заповнення кольорами та диференціації країв. The SC\eval.cpp, представлений на темному тлі, є основним кодом, який реалізує eval функціональність. Пунктирні краї

представляють нерелевантні залежності основного коду від інших програмних файлів після нарізки концепції. Коментарі в програмі були написані чеською мовою, тому для кращого розуміння програми використовувався перекладач Google для перекладу коментарів.

Таблиця 2.4

Показники програми

Attributes	**Count**
#Classes	12
#Class & header Files	21
#Methods	84
#Library Units	114
#Lines of Code	2196
#Blank Lines	236
#Comment Lines	223
#Ratio Comment/Code	0.13

The «Порівняння цілей», «Визначити деталізацію послуги», «Установити пріоритет послуг», і «Відобразити існуючу третю сторону» діяльність не була актуальною в цьому прикладі. Досліджуючи код вручну, специфікації служби були задокументовані шляхом визначення функції, яка обчислює вираз, і необхідних параметрів. Щоб полегшити «Побудова та тестування сервісу» фазі, CodeSurfer [11], інструмент нарізки C/C++, використовувався для вилучення застарілого коду. Видобутий код спочатку протестовано у версії Visual Studio 2010 Ultimate та файлі спільної бібліотеки, evaluate.dll, створюється. Щоб розгорнути службу та зробити її доступною для клієнтів, файл дескриптора служби, service.xml, створюється. У лістингу 2.1 зображено оцінити інтерфейс оцінити веб-сервіс. Нарешті, service.xml і evaluate.dll розгортаються у структурі веб-сервісу WSO2/C++ [29]. Сервіс успішно протестований шляхом розробки клієнтського додатку.

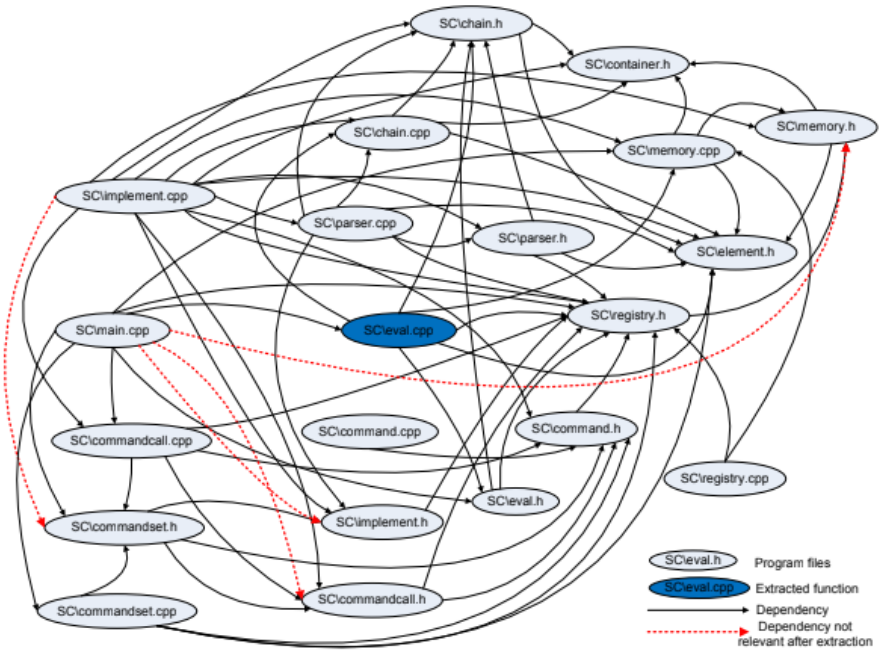

Рисунок 2.5 - Граф залежностей

Метод serviciFi було успішно оцінено та доведено, що він здійсненний і практичний у двох прикладах. Крім того, результати цих тематичних досліджень також доповнили деякі показники якості та висновки експертних оглядів. Успішне вилучення послуг в обох прикладах підтвердило застосовність, ефективність і надійність методу serviciFi. Проте міри якості повноти та узгодженості ще належить визначити, оскільки деякі дії на різних етапах методу serviciFi були пропущені. Крім того, вилучення послуг як ітераційний процес не було застосовним, оскільки обидва тематичні дослідження стосувалися лише одного вилучення служби. Проте досвід вилучення послуг у тематичних дослідженнях доповнили огляд експертів щодо показників ефективності та надійності якості. Як показує аналіз експертних відгуків, на ефективність і надійність методу впливає наявність інструментів і технік, а також характеристики застарілої системи, що було відображено під час проведення кейсів. Значний час було витрачено на виконання ручного нарізання в прикладі COBOL, тоді як у прикладі C++ це було легко завдяки наявності інструменту нарізки. Крім того, доступність оригінального програміста та його підтримка покращили процес вилучення послуг у прикладі COBOL.

```
<service name="evaluate">
  <parameter name="ServiceClass"
        locked="xsd:false">evaluate</parameter>
  <description>
    The Calculator evaluation function presented as Evaluate service
  </description>
  <operation name="evaluate">
  <messageReceiver class="wsf_cpp_msg_recv"/>
  </operation>
</service>
```

Рисунок 2.6- Граф залежностей

Нижче наведено основні загрози достовірності результатів дослідження [29]: (i) достовірність надійності, (ii) внутрішня достовірність і (iii) зовнішня достовірність. Достовірність надійності стосується повторюваності (тобто, якщо той самий випадок виконується іншим дослідником пізніше, дотримуючись тих самих процедур, які проводив попередній дослідник, результат останнього також повинен отримати ті самі знахідки та висновки). Що стосується надійності, повторення промислового прикладу на основі COBOL було неможливим. Але процедури, дотримувані в початковому прикладі, добре задокументовані, і ті самі процедури були дотримані в прикладі C++, який проводив інший дослідник. Потенційною загрозою для внутрішньої валідності була пряма участь авторів в обох тематичних дослідженнях, що може внести упередженість у результат. Щоб мінімізувати загрозу для внутрішньої валідності, два тематичні дослідження були проведені двома різними дослідниками, а результат проаналізував третій дослідник. Що стосується зовнішньої валідності, необхідно буде провести більше тематичних досліджень, щоб розширити результати поточних тематичних досліджень. Різний контекст виконання двох тематичних досліджень (тобто, один промисловий, а інший експериментальний) доповнив можливість узагальнення

2.5 Висновки по розділу

У цьому розділі було представлено метод serviciFi для міграції із застарілої версії на SOA, який було розроблено за допомогою розробки методів і нарізки концепції. На відміну від інших підходів, про які повідомляється в літературі, метод serviciFi успішно поєднав можливість міграції з допоміжною технологією, щоб виставити застарілий код як послуги. Метод serviciFi було оцінено, покращено за допомогою оглядів експертів і доведено, що він можливий для міграції застарілих систем на SOA за допомогою двох прикладів. Основою методу є розробка методу, яка повторно використовує фрагменти методу з існуючих методів сервіс-орієнтованої розробки та вдосконалена методом нарізки концепції для розробки послуг шляхом вилучення застарілого коду.

РОЗДІЛ 3. ВПЛИВ ЕВОЛЮЦІЇ СЕРВІСНО-ОРІЄНТОВАНОЇ АРХІТЕКТУРИ НА МОДЕРНІЗАЦІЮ ЗАСТАРІЛИХ СИСТЕМ: СИСТЕМАТИЧНИЙ ОГЛЯД

3.1 Передумови та актуальність теми модернізації через SOA

Останнім часом багато підприємств зосередилися на підвищенні гнучкості свого бізнесу та досягненні співпраці між підприємствами, щоб залишатися конкурентоспроможними на ринку та досягати своїх бізнес-цілей. Підприємства особливо стикаються з постійними змінами в бізнес-середовищі та змінами в допоміжних інфраструктурах інформаційних технологій (IT), які перешкоджають загальному успіху підприємств [28]. Крім того, більшість підприємств все ще покладаються на так зване застаріле системне програмне забезпечення, розроблене протягом попередніх десятиліть з використанням мов програмування 3GL, таких як COBOL, RPG, PL/I, C, C++. Незважаючи на добре відомі недоліки, такі як негнучкість і складність обслуговування, застарілі системи все ще життєво важливі для підприємств, оскільки вони підтримують складні основні бізнес-процеси; їх неможливо просто видалити, оскільки вони реалізують і зберігають критичну бізнес-логіку. Не дивно, що знання, які містяться в цих системах, мають високу цінність для підприємства. З іншого боку, недостатньо належної документації, кваліфікованої робочої сили та ресурсів для розвитку цих застарілих систем. Таким чином, зростає імпульс до розвитку та повторного використання цих застарілих систем у нових технологічних середовищах, причому Сервісно-орієнтована архітектура (SOA) є найбільш перспективною [32, 17].

SOA виникла як архітектурний стиль, який дозволяє повторно використовувати існуючі застарілі активи в рамках нової парадигми, яка сприяє слабкому зв’язку, абстракції основної логіки, гнучкості, повторного використання та можливості виявлення [20]. Перехід від застарілої системи до SOA може бути вигідним як з економічної, так і з технічної точки зору. З економічної точки зору, еволюція спадщини SOA сприяє управлінню змінами, включаючи зміни всередині організації та зміни на підприємствах [14, 15, 21]. З технічної точки зору безперебійне співробітництво підприємства через створення сервісів [14] і зниження вартості обслуговування вважаються довгостроковими перевагами [21 28]. Враховуючи ці переваги, було проведено

значні дослідження еволюції SOA. Однак немає систематичного огляду еволюції застарілих систем до SOA, особливо зосереджуючись на техніках, методах і підходах, що використовуються для розвитку застарілих систем до середовища SOA. У систематичному огляді літератури, проведеному Razavian & Lago [22], наведено огляд родин міграції SOA. Він зосереджений на класифікації підходів до міграції SOA на вісім різних груп. Класифікація натхненна методом підкови реінжинірингу [25], а не дає історичний огляд методів міграції SOA. Крім того, короткий огляд еволюції спадщини SOA наведено Almonaies et al. [8], який поділяє підходи до еволюції SOA на чотири категорії: заміна, перепланування, упаковка та міграція. Підходи до еволюції спадщини SOA, про які повідомлялося в цьому дослідженні, не базувалися на жодному систематичному процесі огляду літератури, тому повного історичного огляду підходів до еволюції SOA все ще бракує.

У цьому розділі ми надаємо систематичний огляд літератури (SLR) існуючої літератури, що стосується еволюції SOA. Ми надаємо історичний огляд спадщини підходів до еволюції SOA, про які повідомляють наукові кола. Ми зосереджуємось на виявленні технік, методів і підходів, які мають відношення до еволюції спадщини на SOA або які сприяють процесу еволюції спадщини на SOA. Щоб надати такий історичний огляд, ми розробили структуру оцінки, натхненну трьома структурами еволюції програмного забезпечення, про які повідомляється в літературі. Структура оцінювання складається з шести окремих етапів, і кожен етап має власні критерії оцінки будь-якого спадкового підходу до розвитку SOA, про який повідомляють наукові кола. Основні внески цього дослідження такі:

- Історичний огляд еволюції спадщини SOA.
- Успадкована структура процесу еволюції SOA.
- Перелік методів і технік, що використовуються на різних етапах еволюції спадщини SOA.
- Серія дослідницьких проблем і рекомендацій для майбутніх напрямків досліджень.

Ми стверджуємо, що наша система оцінювання дозволяє більш повно зрозуміти еволюцію спадщини SOA, дозволяючи нам визнати внесок, зроблений на даний момент, можливості для комбінування підходів і виявлення відкритих питань і дослідницьких проблем, які все ще існують у спадщині еволюції SOA. Ми вважаємо, що такий огляд принесе користь академічним дослідникам і практикам промисловості. Академічні дослідники можуть стежити за визначеними дослідницькими проблемами, щоб сприяти еволюції SOA, тоді як практикуючі промисловці можуть застосовувати різні методи та техніки, про які повідомляється в дослідженнях у реальних промислових практиках. Глава структурована таким чином: Розділ 3.2 містить деталі нашого методу дослідження; Розділ 3.3 представляє структуру оцінювання; Розділ 3.4 обговорює огляд первинних досліджень; Розділ 3.5 детально описує висновки нашого SLR, Розділ 3.6 обговорює висновки та найкращі методи розвитку успадкованої SOA та описує загрози валідності. Нарешті, у розділі 3.7 ми представляємо висновки нашого дослідження та можливі майбутні напрямки досліджень.

3.2 Метод дослідження

Ми прийняли процедури проведення систематичного процесу огляду на основі керівних принципів, запропонованих [15]. Систематичний огляд складається з протоколу огляду, який детально описує обґрунтування опитування, цілі дослідження, стратегію пошуку, критерії відбору, вилучення даних, синтез та аналіз вилучених даних та інтерпретацію результатів. Такий процес перегляду, як правило, доречний у нашому дослідженні, оскільки він узагальнює наявні внески, визначає прогалини в поточному дослідженні та шляхи для подальших досліджень, а також забезпечує основу для позиціонування нових дослідницьких заходів у рамках дослідження.

Протокол огляду — це план, у якому визначено процедури, які необхідно виконати перед виконанням систематичного огляду. Такий протокол перегляду

описує, як проводити пошук, відбирати відповідні дослідження та критерії відбору, а також аналіз витягнутих даних. Протокол огляду складається з наступного: питання дослідження, джерела даних, стратегія пошуку, стратегія вибору дослідження, вилучення даних та синтез даних. Перші чотири визначають обсяг і мотивацію дослідження, тоді як останні два описують, як на основі даних виводяться результати.

Щоб досягти нашої мети створити огляд підходів до еволюції SOA, ми сформулювали наступні питання дослідження:

1. Як можна систематично визначити метод еволюції спадщини SOA?

2. Які методи та прийоми використовуються для полегшення такого систематичного переходу до методу еволюції SOA?

3. Які існують проблеми дослідження та якою має бути програма майбутніх досліджень у спадщині від еволюції SOA?

Для нашого дослідження ми включили такі вісім електронних бібліотек/джерел індексування як джерела даних: ACM Digital Library, CiteseerX, IEEE Xplore, ISI Web of Knowledge, ScienceDirect, Scopus, SpringerLink і Wiley Inter Science Journal Finder.

Ми створили пошуковий рядок, використовуючи SOA, legacy та migration як основні ключові слова, а також включили синоніми та пов’язані терміни. Після цього рядок пошуку створюється за допомогою логічного «І» для з’єднання трьох ключових слів і логічного «АБО», щоб дозволити синоніми та варіанти класів слів для кожного ключового слова. Отриманий пошуковий рядок зображено в лістингу 3.1.

```
(SOA OR "Service-Oriented" OR "Service-Based" OR "Service-Centric" OR "Service-Engineering" OR "SOSE"
    OR "web service" OR "service-oriented computing") AND(Monolith OR "legacy code" OR "Legacy system"
    OR "existing system" OR "legacy component" OR "legacy software" OR "monolithic system" OR "existing
    software" OR "pre-existing software" OR "legacy information system" OR "legacy program" OR "pre-existing
     assets") AND(migration OR evolution OR modernisation OR reengineering OR re-engineering OR reuse OR "
    service identification" OR "candidate service identification" OR "service extraction" OR bridging OR
    reconstruction OR modernization OR decomposing OR "incubating services" OR integrating OR redesigning OR
     "Service mining" OR migrating OR transformation)
```

Лістинг 3.1: Пошуковий рядок

Рядок пошуку було виконано в цифрових бібліотеках/службах індексування заголовків, рефератів і метаданих – припускаючи, що вони забезпечують стислий підсумок роботи. Окрім рядка пошуку, у стратегії пошуку також має бути визначено діапазон дат навчання. Ми вирішили вибрати 2000 рік як початковий рік для стратегії пошуку, оскільки SOAP [38] вперше було подано до W3C у 2000 році.

Цілком імовірно, що деякі результати (дані дослідження) пошуку можуть містити ключові слова, але не мають відношення до нашого дослідження. Наприклад, дані дослідження під назвою «Оцінка застарілих систем і систем грід-сервісів сфери охорони здоров'я: початковий крок до трансформації до хмарної системи» включені в результат початкового відбору. Для того, щоб виключити такі нерелевантні дослідження, відбір досліджень виконується так, що дані дослідження оцінюються для визначення фактичної відповідності. Нами було визначено набір критеріїв включення та виключення на основі обсягу дослідження та якості досліджень. Критерії включення та виключення наведені в таблиці 3.1.

Вибір дослідження не тільки усуває нерелевантні дослідження, але й забезпечує якість дослідження та обсяг дослідження. Наприклад, критерій включення I1 і критерій виключення E4 гарантують, що дані дослідження відповідають стандартам рецензованих наукових статей. Критерії включення I2, I3 та критерії виключення E1, E2 та E3 охоплюють дослідження відповідно до мети/мотивації дослідження.

Таблиця 3.1

Критерії включення та виключення для відбору дослідження

Inclusion Criteria	Exclusion criteria
I1. A study in the form of a scientific peer-reviewed paper. **Motivation:** A scientific paper guarantees a certain level of quality through a peer review process and contains a substantial amount of content.	E1. A study that is not about legacy to SOA evolution. **Rationale:** Our objective is to study legacy to SOA evolution, so we exclude any other legacy modernization. For example, legacy modernization to object-orientation, cloud computing or grid services will be excluded.
I2. A study that is focused on legacy to SOA evolution. **Motivation:** We are interested in legacy to SOA evolution, which implies that any study targeting legacy to SOA evolution should be included.	E2. A study that is related to challenges and issues while modernizing legacy systems to SOA. **Rationale:** We focus on a specific solution to legacy to SOA evolution. We exclude papers with an objective of presenting challenges, issues, and future directions to legacy to SOA evolution.
I3. The objective of the study is to present/propose a solution(s) to legacy to SOA evolution. **Motivation:** We are interested in a specific solution to legacy to SOA modernization. A solution could be a complete evolution process/method or solution enabling legacy to SOA evolution.	E3. A study that has other objective(s) than providing a solution(s) to legacy to SOA evolution. **Rationale:** We exclude papers with a main objective other than proposing a solution to legacy to SOA evolution. For instance, we exclude papers with an objective of presenting challenges, issues, future directions to legacy to SOA evolution and comparing the modernization techniques of legacy to SOA evolution.
	E4. The Study is reported in another language than English. **Rationale:** We exclude the papers that are written in languages other than English, since English is the common language for reporting in most of the international venues of computer science

Ми витягли вибірку дослідження в електронну таблицю, включаючи такі деталі: назва, автори, рік публікації, форма публікації (журнал/конференція/практикум/розділ книги), назва та анотація. Ми провели перший раунд відбору на основі «назви та анотації» дослідження. Дослідження було класифіковано таким чином: (i) релевантне (дослідження в межах дослідження), (ii) нерелевантне (дослідження поза межами дослідження) і (iii) помірне (неможливо визначити релевантність статті). Для кожного нерелевантного та помірного дослідження в електронній таблиці наводилися чіткі причини. Помірна категорія була визначена шляхом повторного рецензування рецензентом, відмінним від початкового рецензента, та шляхом обговорення статті з командою. Кінцевим результатом є збір відповідних досліджень, які ми називаємо первинними дослідженнями.

Первинні дослідження були оцінені за системою оцінки, і результати представлені в наступних розділах. Ми провели процес перевірки, дотримуючись протоколу перевірки. Спочатку ми мали 8493 звернення, коли ми запускали пошуковий запит по електронним бібліотекам/джерелам індексування. Ці 8493 статті були проаналізовані п’ятьма дослідниками, щоб визначити релевантність на основі назви та анотації, що залишило 269 статей.

Потім ці статті були оцінені на основі критеріїв включення та виключення, що призвело до 121 первинного дослідження. Подробиці процесу перегляду можна знайти в Idu et al. [144]. На рисунку 3.1 зображено процес перевірки.

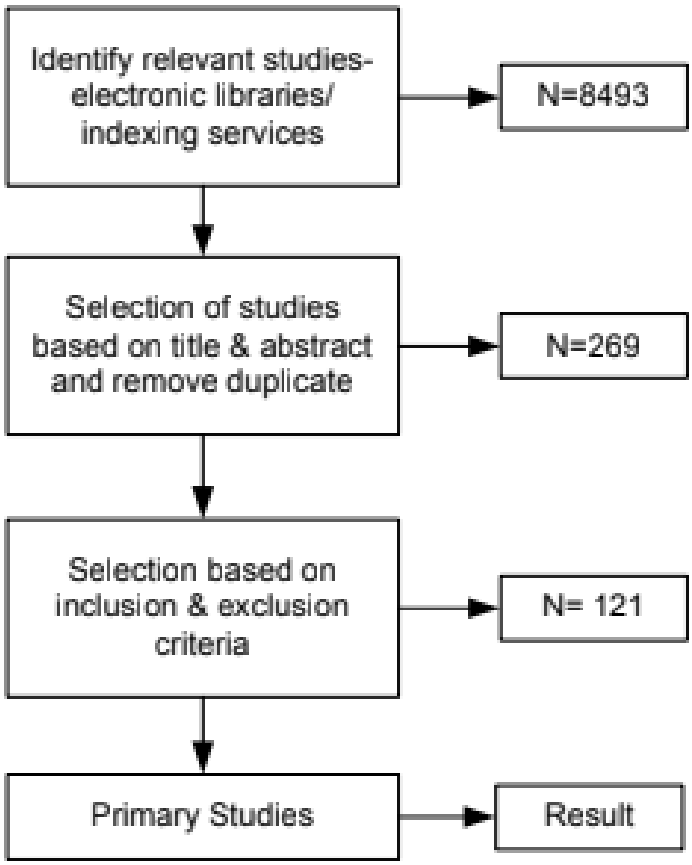

Рисунок 3.1 - Процес перевірки з кількістю досліджень

3.3 Структура оцінки еволюції від спадщини до SOA

Щоб розробити структуру оцінки еволюції застарілої системи до SOA, нам потрібно було визначити фази, які зазвичай пов’язані з еволюцією/модернізацією застарілих систем. Завдяки високій кількості цитувань (популярності), наявності документації та повноті процесу еволюції/модернізації застарілої версії, для визначення фаз нашої системи

оцінювання були використані наступні методи з області реінжинірингу програмного забезпечення: метод метелика [29], метод Відродження [28] та модернізація, керована архітектурою (ADM) [15]. Основною причиною використання цих методів еволюції/модернізації є те, що область реінжинірингу

програмного забезпечення була широко досліджена та широко практикується в промисловості порівняно з методами еволюції SOA. Зокрема, ми хочемо повторно використати концепції цих методів у розробці нового методу для еволюції спадщини до SOA.

Інженерія методів [3] дозволяє нам повторно використовувати існуючі концепції з існуючих методів для створення нових методів. Тому ми використовуємо розробку методів і повторно використовуємо концепції з трьох вищезазначених методів еволюції/модернізації. Ми стверджуємо, що повторне використання методів і практик із існуючих стандартів/методів економить час і зменшує проблему впровадження (тобто легше адаптуватися до існуючих методів/практик, ніж вивчати нові методи). Подробиці побудови системи оцінки описані в Idu et al. [14]. Можна стверджувати, що існує достатня відповідна спадщина методів еволюції SOA, які можна було б використати для розробки структури оцінювання. Більшість методів еволюції спадщини SOA, про які повідомляється в літературі, або зосереджуються на розробці допоміжних технологій (тобто методів реалізації для виявлення застарілих систем у SOA), або на плануванні еволюції спадщини SOA (тобто на визначенні можливості еволюції) [22].

Однак спадщина еволюції SOA вимагає консолідації обох, розробки допоміжних технологій і планування спадщини еволюції SOA [7, 14]. У нашому підході ми прагнемо розробити таку структуру, яка поєднує обидва аспекти (тобто планування еволюції та впровадження SOA у спадок). Крім того, ми прагнемо оцінити ці існуючі успадковані методи розвитку SOA за допомогою нашого розробленого методу оцінки, а не використовувати їх для розробки нового методу. З трьох методів ми визначили фази, спільні для всіх них. Наприклад, розуміння застарілої системи, розуміння цільової системи, визначення доцільності еволюції і впровадження еволюції є загальними фазами у вищезгаданих методах.

Щоб зробити нашу структуру оцінки більш актуальною для домену SOA та відобразити намір спадщини щодо розвитку SOA, ми додатково

проаналізували та визначили деякі етапи з наступних методів сервіс-орієнтованої розробки: сервіс-орієнтована методологія проектування та розробки (SODDM) [20], методологія впровадження веб-сервісу (WSIM) [16] та сервіс-орієнтоване моделювання та архітектура (SOMA) [12]. Деталі ідентифікації фаз детально описані Reijnders et al. [22] з використанням методичного інженерного підходу. З цих методів розробки, орієнтованих на послуги, ми додали до нашої системи оцінки етапи ідентифікації потенційних служб, розгортання й надання.

Нарешті, наша система оцінювання включає шість фаз, розділених на два загальні етапи. Структура оцінювання та етапи зображені на рисунку 3.2. Планування еволюції стосується питання «що робити?» і «чи можлива еволюція в даному контексті?» Впровадження та управління еволюцією відповідає на питання «як це зробити?» і «які техніки можна використовувати для виконання еволюції?» У наступних підрозділах ми пояснюємо етапи нашої системи оцінювання.

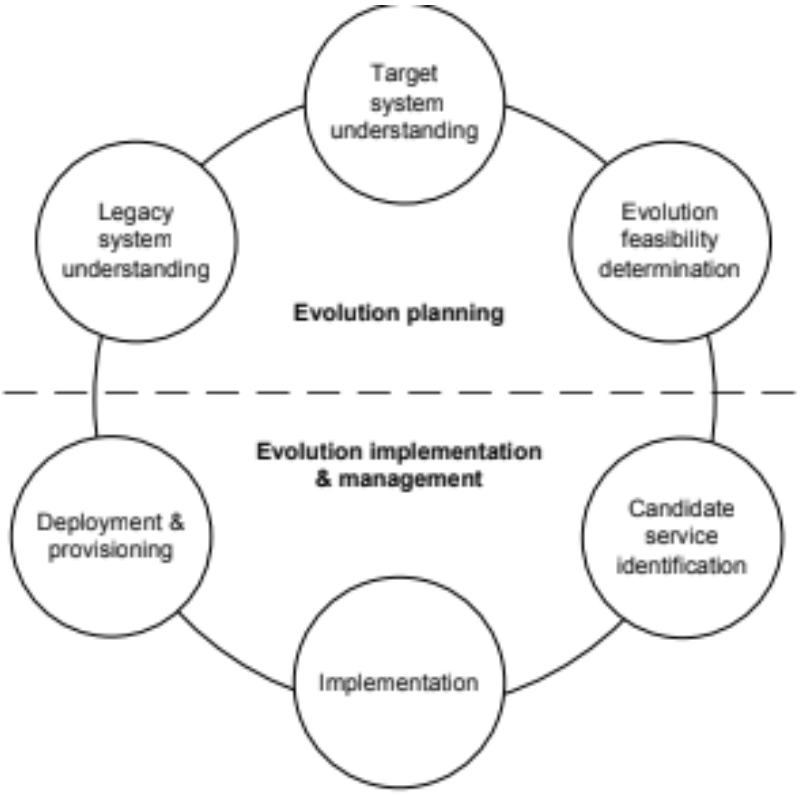

Рисунок 3.2 - Структура оцінювання

Розуміння успадкованої системи та її ситуації «як є» має вирішальне значення для успіху будь-якої еволюції [24]. Це включає в себе детальний аналіз успадкованої системи та використання різних методів. Наприклад, можна використовувати зворотне проектування, розуміння програми, відновлення архітектури, часто з підтримкою інструментів для створення системних артефактів. Розуміння застарілої системи часто включає аналіз історії розробки, опитування розробників (якщо такі є) і поточних користувачів, щоб прийти до розуміння архітектури застарілої системи. У нашій структурі оцінки ми визначили критерії оцінки, щоб дослідити, чи включає будь-який метод еволюції спадщини до SOA розуміння застарілої системи та якою мірою цей етап обговорюється.

Фаза розуміння цільової системи полегшує представлення бажаної архітектури SOA, яка буде бути. На цьому етапі описується цільове середовище SOA, яке включає такі дії, як визначення основних компоненти/функціональні можливості середовища SOA, конкретні технології та стандарти, які будуть використовуватися, стан цільової SOA та доступність існуючих подібних служб для повторного використання. У нашій структурі оцінки ми визначили критерії оцінки, щоб визначити, чи включає метод еволюції спадщини SOA розуміння цільової системи для бажаної системи SOA, і в якій мірі цей етап обговорюється.

Фази розуміння успадкованої системи та розуміння цільової системи забезпечують краще розуміння ситуацій, які є та які мають бути, відповідно. На основі цього розуміння необхідно визначити здійсненність еволюції, і це робиться на етапі визначення здійсненності еволюції. Оцінки здійсненності виконуються на технічному, економічному та організаційному рівнях. Технічна оцінка включає вимірювання складності коду застарілої системи з точки зору згуртованості, зчеплення, повторного використання та абстракції [27]. Економічна оцінка включає визначення економічної доцільності еволюції, наприклад, за допомогою аналізу витрат і вигод, як запропоновано Снідом [26]. Після аналізу технічної та економічної здійсненності організація схвалює проект еволюції, також враховуючи, чи відповідає передбачувана система SOA її бізнес-

цілям. У нашій структурі оцінки ми визначили критерії оцінки, щоб визначити, чи включає метод еволюції, що перейшов у спадщину SOA, можливість еволюції, і якщо так, то як це виконується.

Застарілі системи піддаються еволюційному розвитку та виправленню помилок у вихідному коді часто людьми, які не розробляли його. Зазвичай це призводить до великої надмірності в коді. Крім того, погана документація та відсутність відповідних ресурсів (наприклад, розробників, архітекторів) ускладнюють розуміння вихідного коду. У такому сценарії ідентифікація потенційних сервісів і багатих сервісами областей у застарілому коді, безумовно, є складним завданням [14]. Фаза ідентифікації потенційних послуг спрямована на визначення територій, багатих послугами. Для цього можна використовувати різні техніки. Наприклад, архітектурна реконструкція, розташування функцій, відновлення шаблону проектування, методи кластерного аналізу, аналіз концепції та візуалізація вихідного коду можуть бути використані для ідентифікації багатих послугами областей у великій частині застарілого коду. У нашій системі оцінки ми визначили критерії оцінки, щоб дослідити, чи містить будь-який застарілий метод еволюції SOA методи визначення потенційних потенційних послуг.

Етап впровадження стосується технічної еволюції всієї застарілої системи до цільової системи з використанням різних методів, які часто підтримуються інструментами. Наприклад, обгортання, нарізка програми, нарізка концепції, перетворення графа, переклад коду, перетворення програми на основі моделі, сканування екрана, технологія кодових запитів і перетворення графа можуть використовуватися для отримання/використання застарілого коду як послуг. У нашій структурі оцінки ми визначили критерії оцінки, щоб дослідити, чи метод еволюції від успадкованої системи до SOA включає будь-які методи вилучення/використання успадкованого коду як послуг.

Етап розгортання та надання пов’язаний із розгортанням та керуванням службами після вилучення застарілого коду. Після вилучення сервіси розгортаються в сервісній інфраструктурі. Надання послуг зазвичай включає дії

після розгортання, такі як публікація, версія служб, вимірювання та виставлення рахунків за використання послуг [16]. У нашій структурі оцінки ми визначили критерії оцінки, щоб визначити, чи включає в себе розгортання та ініціалізацію застарілий метод еволюції SOA. На основі визначених етапів ми склали список критеріїв оцінювання, наведений у таблиці 3.2: перший стовпець представляє етапи в рамках еволюції, другий стовпець перераховує визначені етапи нашої системи оцінювання, третій стовпець представляє питання оцінювання як критерій оцінювання для кожного етапу, а останній стовпець містить можливі відповіді на кожне запитання оцінювання. Відповіді можуть бути трьох типів: Так/Ні – для вказівки на відповідність заданому критерію, розповідь – для відповіді на відкрите запитання та шкала – для кількісного визначення ступеня підтримки будь-якого критерію. Оцінка масштабу представлена в таблиці 3.3.

Таблиця 3.2

Критерії оцінювання на основі системи оцінювання

Stage	Phase	Evaluation Criteria	Answer
Evolution Planning	Legacy System Understanding	Does the solution include legacy system understanding?	Yes/No
		Which technique(s) is used for legacy system understanding?	Narrative
		To what extent are those techniques used?	Scale
		Is there any tool support for legacy system understanding?	Yes/No
	Target System Understanding	Does the solution include target system understanding?	Yes/No
		What criteria/factors are included for target system understanding?	Narrative
		To what extent are those criteria/factors used?	Scale
	Evolution Feasibility Determination	Does the solution include evolution feasibility assessment?	Yes/No
		What technique(s) is used for evolution feasibility assessment?	Narrative
Evolution Implementation & Management	Candidate Service Identification	Does the solution include candidate service identification?	Yes/No
		What technique(s) is used for identifying candidate services?	Narrative
		Is there tool support for candidate service identification?	Yes/No
	Implementation	Does the solution provide any implementation technique for evolution?	Yes/No
		What technique(s) is used for implementation?	Narrative
		Is there tool support for the implementation?	Yes/No
	Deployment & Provisioning	Does the solution provide deployment & provisioning of the services?	Yes/No
Case Study		What empirical evidence (industrial/experiment) is provided?	Narrative
		In which language is the legacy system developed?	Narrative

Таблиця 3.3

Шкала суджень для оцінки підтримки використаних технік і методів

Scale point	Scale Definition	Representation
No support	The specified technique is not mentioned.	-
Implicitly discussed	The specified technique is mentioned.	+
Explicitly discussed	The specified technique is mentioned and discussed but no detailed information is given.	++
Explicitly discussed with evidence of use	The specified technique is mentioned, discussed and there is empirical evidence of its usability.	+++

3.4 Огляд первинних досліджень

Загалом виявили 121 публікацію як наші основні дослідження після оцінки за критеріями включення та виключення. На рисунку 3.3 показано розподіл первинних досліджень, опублікованих за рік, разом із лінією тренду. Позитивний нахил лінії тренду не тільки вказує на збільшення обсягу досліджень, що проводяться в області еволюції SOA, але також відображає збільшення підходів до еволюції SOA разом зі зрілістю парадигми SOA – SOA використовується як архітектурний стиль після того, як SOAP [38] було вперше подано W3C у 2000 році. Ми не можемо бути впевнені, що ми охопили всі дослідження з датою публікації в 2011, оскільки на той час навчання ще не було проіндексовано. Це одна з можливих причин різкого скорочення публікацій у 2011 році.

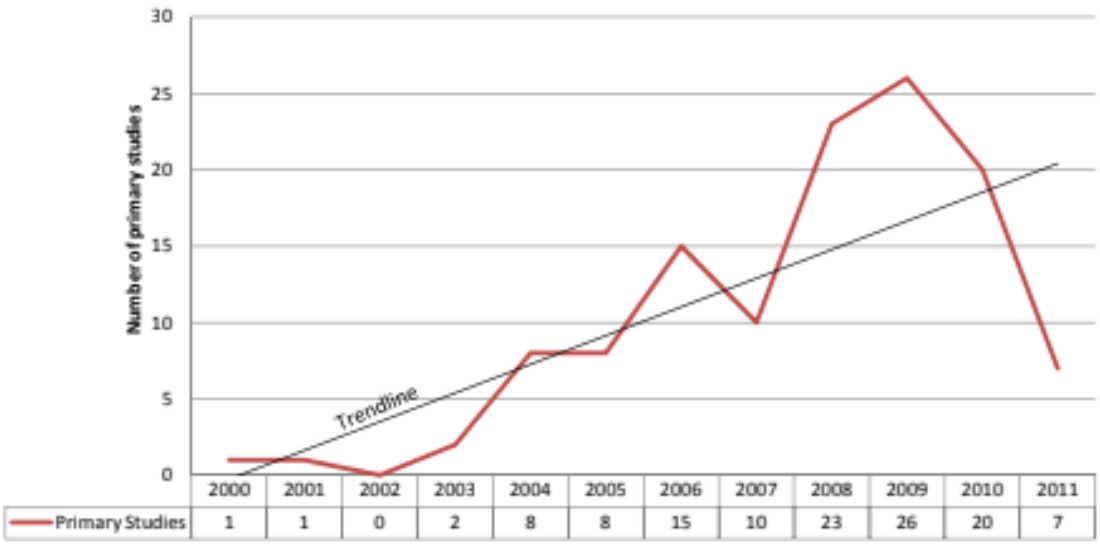

Рисунок 3.3 - Розподіл первинних досліджень, опублікованих за рік

На рисунку 3.4 представлено розподіл первинних досліджень за місцями, з яких було вибрано принаймні дві статті. Дуже цікаво помітити, що найбільша кількість досліджень проводиться на майданчиках, пов'язаних із обслуговуванням, еволюцією та реінжинірингом систем, таких як CSMR, ICSM, WSE, а не на основних обчислювальних майданчиках, орієнтованих на послуги, таких як SCC, ECOWS, ICSOC. Це означає, що еволюція спадщини SOA часто розглядається як вирішення проблем підтримки/еволюції (застарілих) програмних систем. Крім того, частота публікацій у журналах відносно низька порівняно з конференціями чи семінарами, що не дивно в такій молодій галузі. Зауважте, що ми не включили місця проведення менше двох разів.

У таблиці 3.4 представлено розподіл первинних досліджень за типом джерела. Результати показують, що конференції є найбільш широко використовуваним методом розповсюдження підходів, що відносяться до еволюції SOA. Кількість журнальних статей щодо успадкованих підходів до еволюції SOA менша порівняно з матеріалами конференцій.

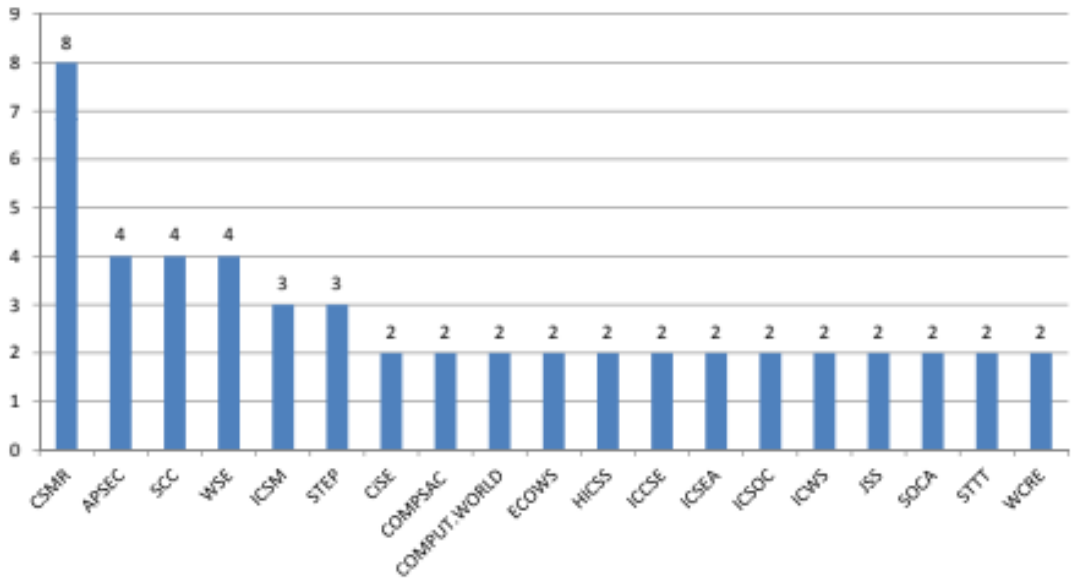

Рисунок 3.4 - Резюме первинних досліджень у різних місцях

Таблиця 3.4

Зведення первинних досліджень за джерелами

Source/Year	00	01	02	03	04	05	06	07	08	09	10	11	Total
Book	0	0	0	0	0	0	0	0	1	2	1	1	4
Conference	1	1	0	1	6	4	13	9	12	17	17	4	85
Journal	0	0	0	0	1	0	2	1	7	5	2	2	20
Workshop	0	0	0	1	1	4	0	0	3	2	1	0	12
Total	1	1	0	2	8	8	15	10	23	26	20	7	121

3.5 Оцінка ефективності підходів до модернізації Legacy-систем

Результат нашого SLR базується на критеріях оцінки, описаних у таблиці 3.2. За нашими критеріями оцінювання ми оцінили 121 видання. Через обмеження місця ми не включили повний результат нашої повної оцінки в цей розділ. У Додатку А наведено оцінку невеликої кількості статей. Щоб отримати повний результат оцінки, ми посилаємося на Khadka et al. [14]. Результат головним чином зосереджений на тому, чи підтримує публікація фази нашої системи оцінювання, які методи та технології використовуються (якщо підтримуються), і чи обговорюється будь-яка інструментальна підтримка методів і технік. Крім того, також представлені деталі емпіричних доказів (ситуаційне дослідження), наведених у кожній публікації. Під час нашої оцінки ми створили перелік методів і технік, згаданих у публікації. Ми не робили жодних суб'єктивних припущень для категоризації. Наприклад, у багатьох публікаціях «архітектурне відновлення» та «архітектурна реконструкція» фази розуміння застарілої системи вважаються ідентичними; однак ми не об'єднали їх в одну. Оскільки ми не робимо жодних суб'єктивних припущень, ми вважаємо, що це зменшить упередженість наших висновків. Ми представляємо наші висновки з двох аспектів: (i) ступінь покриття– вказує, які етапи/фази підтримуються первинними дослідженнями та (ii) використовувані методи та прийоми– перелік методів і технік, які зазвичай застосовуються на практиці на кожному етапі.

Зі 121 публікації 12 публікацій повністю висвітлюють етап планування еволюції, тобто 12 публікацій підтримують етапи розуміння застарілої системи,

розуміння цільової системи та визначення здійсненності еволюції. Окремо, на етапі планування еволюції, 66 публікацій підтримують спадщину розуміння системи, 43 публікації підтримують розуміння цільової системи, а 20 публікацій підтримують етап визначення здійсненності еволюції. Подібним чином, 15 публікацій із 121 повністю висвітлюють етап впровадження та управління еволюцією, тобто 15 публікацій підтримують етапи ідентифікації потенційних послуг, впровадження, розгортання та забезпечення. Окремо 59 публікацій підтримують фазу ідентифікації потенційних послуг, 97 підтримують фазу впровадження та 22 підтримують фазу розгортання та надання послуг. Цікаво, що лише 2 публікації [13, 3] підтримують загальні фази нашої спадщини до структури еволюції SOA. У таблиці 3.5 представлено розподіл первинних досліджень за фазами.

Таблиця 3.5

Розподіл первинних досліджень за фазами

Legacy to SOA Evolution 2					
Evolution Planning 12			**Evolution Implementation & Management** 15		
Legacy System Understanding 66	Target System Understanding 43	Evolution Feasibility Determination 20	Candidate Service Identification 59	Implementation 97	Deployment & Provisioning 22

Ми склали перелік методів і прийомів, про які повідомлялося в первинних дослідженнях, і відповідно зобразили їх у вигляді гістограми, по одному для кожного етапу нашої системи оцінювання. Зауважте, що на більшості етапів інформація була недоступною (Н/Д) і що результати, представлені на стовпчастих діаграмах, не містять Н/Д. На малюнку 3.5 зображено методи та техніки, які використовуються на етапі розуміння застарілої системи. Техніка зворотного проектування є, безумовно, найпоширенішою технікою. Документація та інтерв'ю є другою та третьою найбільш використовуваними техніками, за якими йдуть здебільшого аналіз вихідного коду або методи архітектурної реконструкції. На основі критеріїв масштабу (-, +, ++, +++) 22

статті детально обговорювали розуміння застарілої системи з +++, 18 статей з ++ і 20 статей з +. У більшості випадків для розуміння застарілої системи використовувалося кілька методів і технік. Цікавим є те, що більшість методів і прийомів, які використовуються для розуміння застарілої системи, мають технічний характер, наприклад зворотне проектування, відновлення архітектури, розуміння програм. Ручні методи, такі як документування та інтерв'ю, менш поширені, ніж поглиблені описи технічних методів. Однією з причин використання методів і технік такого технічного характеру є те, що застарілі ресурси, такі як документація та

розробники, є дефіцитними – широко виявлену проблему в еволюції застарілих [22, 32]. Крім того, лише 26 із 121 статті обговорюють підтримку інструментів для розуміння застарілої системи. У більшості документів для розуміння застарілої системи об’єднано кілька методів.

На малюнку 3.6 зображено методи та техніки, які використовуються для розуміння цільової системи. Тут найбільш широко використовується вибір конкретної архітектури. Цікаво відзначити, що майже всі випадки на діаграмі є техніками, які фактично представляють технологічні аспекти цільової системи, тоді як лише Інтерв’ю відноситься до процесу (тобто організаційної) точки зору. Лише в документах 13/121 детально обговорюється розуміння цільової системи з +++, у документах 12/121 з ++ і в документах 12/121 з +.

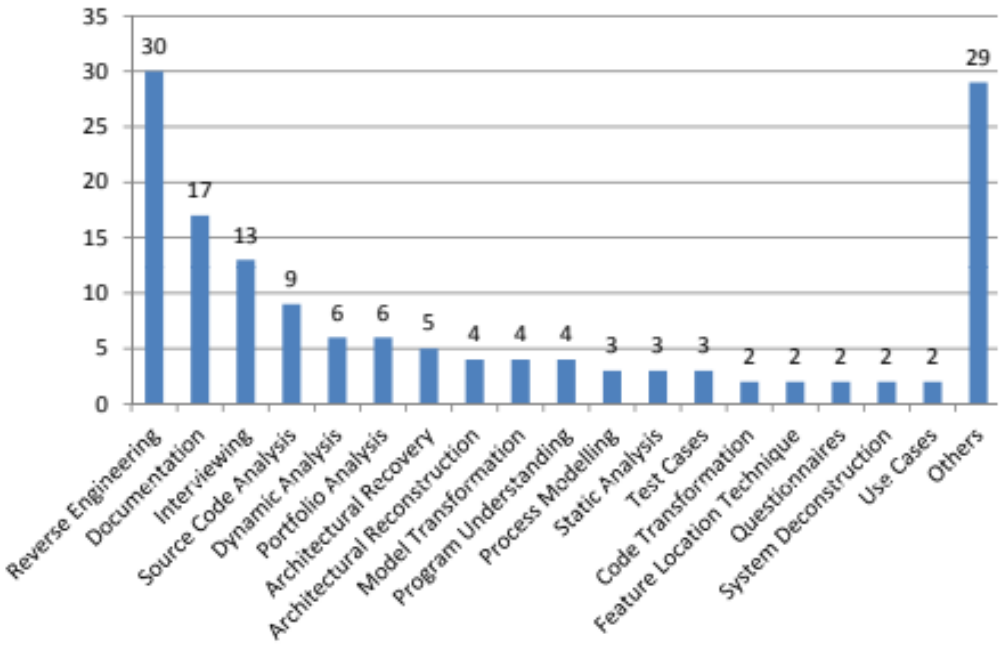

Рисунок 3.5: Розподіл методів і технік, що використовуються для розуміння застарілої системи

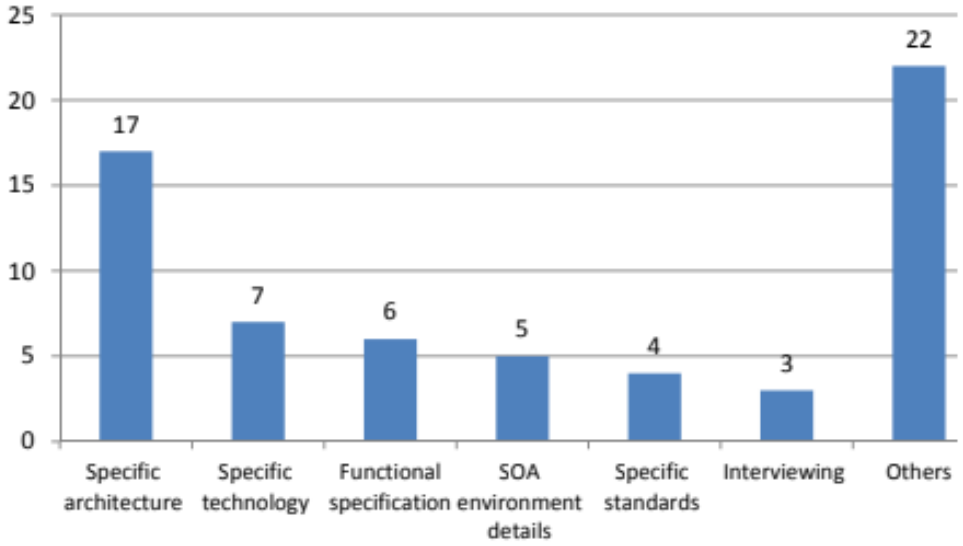

Рисунок 3.6: Розподіл методів і технік, що використовуються для розуміння цільової системи

Методи та прийоми, які використовуються для етапу визначення здійсненності еволюції, показані на рисунку 3.7. Тут широко використовується аналіз витрат і вигод (CBA), за яким слідує складність коду. У той час як метод CBA – це переважно економічно орієнтований аналіз, інші найбільш використовувані методи, оцінка складності коду та багаторазового

використання, відносяться до технічного аналізу. Деталі CBA представлені Sneed [26] та Umar et al. [27], а деталі складності коду пояснює Снід [28]. Концепція аналізу варіантів реінжинірингу (OAR) запропонована Bergey [24]; він використовувався в SMART [15].

На малюнку 3.8 зображено методи та прийоми, які використовуються на етапі ідентифікації послуги-кандидата. Найчастіше використовується ручна ідентифікація. Варто також відзначити, що жодна з інших методик не використовується широко, що призводить до 51 окремої методики, яка зустрічається в первинних дослідженнях, крім ручного. Цікаво відзначити, що ідентифікацію служб-кандидатів також досліджували окремо, щоб сприяти еволюції спадщини SOA.

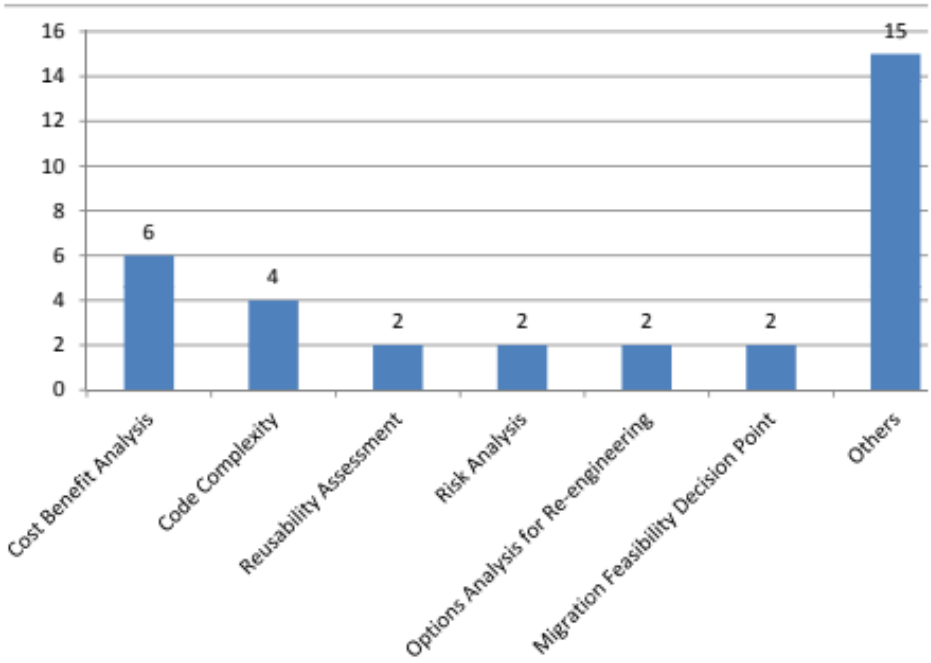

Рисунок 3.7 - Розподіл методів і технік, що використовуються для визначення доцільності еволюції

Алахмарі та ін. [5] пропонують ідентифікацію послуг на основі керованої моделлю архітектури з використанням метамоделі SOA для ідентифікації послуг у застарілому коді. Аверсано та ін. [13] поєднали методи пошуку інформації з метрикою на основі подібності для ідентифікації потенційних послуг у застарілих системах. У [58] автори пропонують підхід, заснований на онтології,

в якому онтологія зберігає знання як про область застосування, так і про застарілий код. Пізніше аналіз формальної концепції та аналіз реляційної концепції використовуються для ідентифікації послуг-кандидатів. Накамура та ін. [197] генерувати діаграми потоку даних із застарілого коду, використовуючи методи зворотного проектування, щоб допомогти ідентифікувати кандидатську послугу. Чжан та ін. [298] використовують техніку кластеризації для аналізу архітектурної інформації та ідентифікації пов'язаних модулів як потенційних потенційних служб.

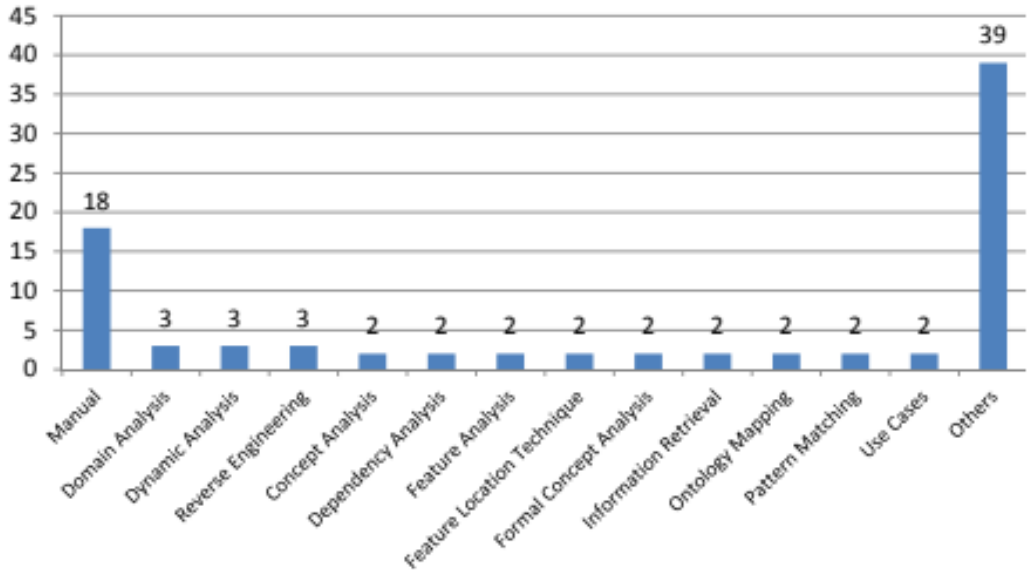

Рисунок 3.8 - Розподіл методів і технік, що використовуються для ідентифікації потенційних послуг

Методи та прийоми, які використовуються на етапі впровадження, представлені на рисунку 3.9. Обгортання є найпоширенішим. Враховуючи велику різницю між Wrapping та іншими використовуваними методами, ми вважаємо, що більшість успадкованих методів розвитку SOA не зосереджені на зміна існуючих застарілих кодових баз. Крім того, обгортання є швидким, менш ризикованим, економічним і простим рішенням, хоча застаріла система залишається монолітною. Результати нашої оцінки показують, що такі методи, як трансформація моделі, нарізка програми та трансформація коду, використовуються набагато рідше. З таблиці 3.5 ми знаходимо, що 97 із 121 документу підтримують фазу впровадження. Під час нашої оцінки ми також

виявили, що 74 документи з цих 97 документів (тобто документи, що підтримують етап впровадження) також мають підтримку інструментів для впровадження. Крім того, 22 публікації підтримують етап розгортання та підготовки.

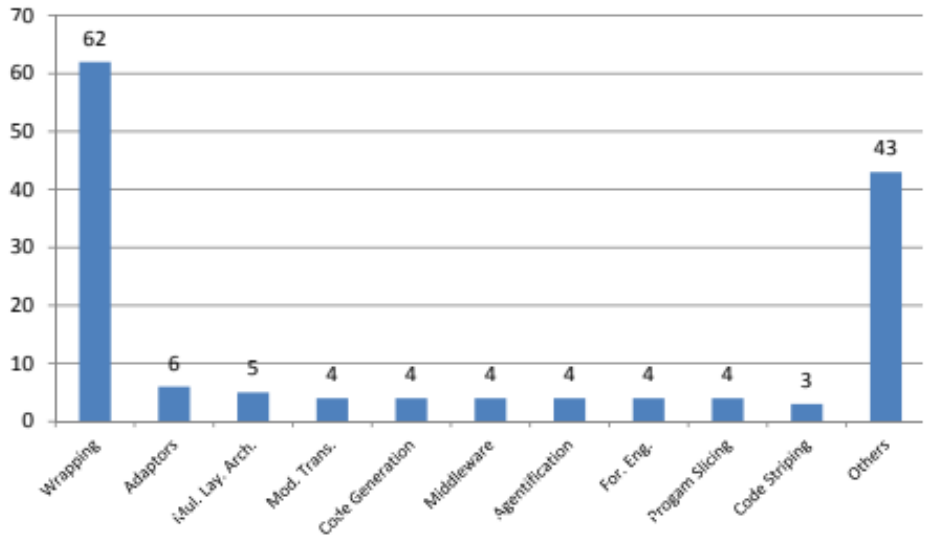

Рисунок 3.9 - Розподіл методів і технік, що використовуються для впровадження

На малюнку 3.10 зображено розподіл емпіричних досліджень, проведених для підтвердження запропонованої еволюції SOA у первинних дослідженнях. Більшість первинних досліджень представляли тематичні дослідження, які проводилися на промисловому рівні. Цікаво відзначити той факт, що була невелика кількість досліджень, які представляли як експериментальні, так і промислові приклади, таким чином охоплюючи ширшу застосовність валідації. Серед промислових прикладів найбільш поширені застарілі системи на основі C++ і COBOL: по чотири випадки для кожної. В експериментальних прикладах широко використовувалися системи на основі Java (всього шістнадцять), за ними йшов COBOL (чотири системи).

Етап планування еволюції системи оцінювання (пор. 3.2) розглядає можливість еволюції з точки зору бізнесу та технічної точки зору. Планування еволюції зосереджується на обґрунтуванні того, чи є успадкована система економічно та технічно придатною для еволюції. Значною мірою успіх і невдача

еволюційного проекту залежить від правильного планування [24]. У контексті переходу від успадкованої системи до SOA планування еволюції стає складнішим, оскільки слід добре розуміти різні технічні фактори успадкованих систем. До таких технічних факторів відносяться показники складності [21] і показники зчеплення та когезії для повторного використання [11, 21]. У випадку застарілих систем отримання такої інформації є складним завданням, особливо через відсутність ресурсів і документації. Інші важливі фактори включають оцінку вартості еволюції та економічну доцільність для визначення прибутковості еволюції. Економічна доцільність повинна брати до уваги поточні витрати на підтримку застарілої системи та прогнозовані витрати на підтримку цільової системи після еволюції. Отже, планування розвитку також має враховувати архітектуру та стандарти цільової системи.

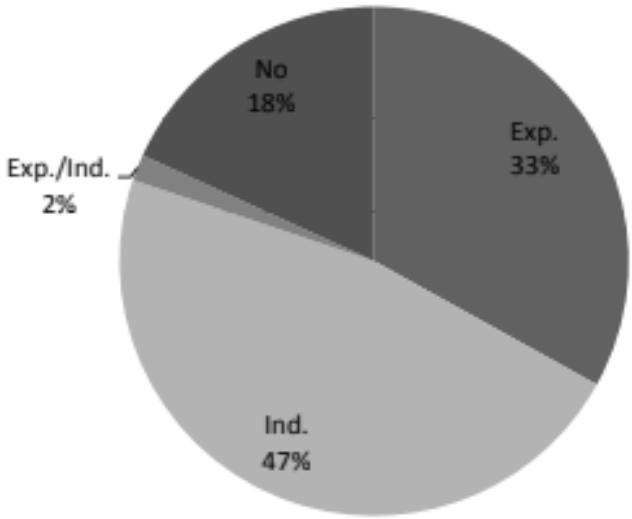

Рисунок 3.10 - Розподіл проведених прикладів

У рамках планування еволюції розуміння застарілої системи було широко досліджено, насамперед за допомогою методів зворотного проектування. Двома основними категоріями зворотного проектування в розумінні застарілої системи є розуміння програми та архітектурна реконструкція. Розуміння програми [70] визначається як процес отримання знань про комп'ютерну програму та широко використовується для підтримки програмного забезпечення, еволюції

програмного забезпечення та реінжинірингу програмного забезпечення. Corbi [70] визначає три дії, які можна використати для розуміння програми: читання про програму (наприклад, документація), читання самої програми (наприклад, читання вихідного коду) і запуск програми (наприклад, спостерігати за виконанням, отримувати дані трасування тощо). Усі ці три дії використовувалися в еволюції SOA в різних пов'язаних темах, таких як документація, аналіз вихідного коду, статичний аналіз і динамічний аналіз. У більшості статей зазначено, що для розуміння застарілих кодів використовуються методи розуміння програми або аналізу вихідного коду. Однак лише кілька статей детально пояснюють такі методи розуміння програм. Методи аналізу вихідного коду були добре представлені Zhang et al. [300]; статичний аналіз у [3] за допомогою маніпулятора потокового графіка (FGM) і динамічний аналіз за допомогою JGrabLab/GReQL у [30] та TGraph від Fuhr er al. [92]. Архітектурна реконструкція — це процес, у якому представлення архітектури програмної системи отримують з існуючого вихідного коду [13] і широко використовується в області реінжинірингу програмного забезпечення. Подібним чином повідомлялося про використання архітектурної реконструкції в підходах до еволюції SOA. Куадрадо та ін. [7] використовували робочий процес QUE-es Architecture Recovery (QAR) для реконструкції архітектури застарілих систем за допомогою Jude, Omondo UML studio та інструменту Eclipse Test and Performance Tools Platform (TPTP). Льюїс та ін. [17] та O'Brien et al. [20] використовував інструмент ARMIN для реконструкції архітектури застарілої програми управління та контролю Міністерства оборони (C2), щоб виявити різні незадокументовані залежності у вихідному коді. Лі та Тахвілдарі [17] використали інструментарій Extracting Business Services (E-BUS) для реконструкції архітектури різних систем на основі Java. Подібним чином Zhang et al. [30] використовують відновлення архітектури для отримання проектної та архітектурної інформації, яка використовується як вхідні дані для ідентифікації послуги. Наша оцінка показує, що архітектурна реконструкція частіше використовується з інструментальною підтримкою, ніж з програмним

розумінням. Крім того, в більшості випадків було використано як програмне розуміння, так і архітектурну реконструкцію. Методи визначення місця розташування ([5, 28]) також були використані для розуміння застарілих систем.

На етапі розуміння цільової системи мається на увазі вибір архітектури та відповідних технологій SOA майбутньої системи, що зрештою відіграє важливу роль у якості майбутньої системи SOA. Льюїс та ін. [17] стверджує, що характеристики цільової системи стримають рішення про те, чи можна повторно використовувати застарілі компоненти. В основному розуміння цільової системи можна розглядати з двох точок зору: функціональні характеристики та технічні характеристики цільової системи. Функціональні характеристики включають потенційні функціональні можливості, які слід розвинути на основі застарілого коду. Цей процес називається проектуванням сервісу. Він також визначає, до якого рівня деталізації мають бути визначені послуги, і, відповідно, оркестровкою служб необхідно керувати для підтримки бізнес-процесів. Слід також враховувати різні функціональні та нефункціональні властивості, такі як ремонтопридатність, сумісність, швидкість реагування, продуктивність, безпека та доступність. Технічні характеристики цільового середовища включають технологію обслуговування (на основі SOAP або REST), технології обміну повідомленнями, протоколи зв’язку, мови опису послуг і механізми виявлення послуг. Незважаючи на важливість, у більшості статей розуміння цільової системи не описано детально. Швидше, у статтях просто зазначено, що цільова архітектура або цільова система є важливим аспектом. Однак функціональні характеристики розуміння цільової системи були добре вивчені в SOAMIG [9] і SMART [17, 17]. Метод SOAMIG описує важливість дизайну сервісу, який є результатом прямого проектування (проектування цільової архітектури та оркестровки сервісів) і зворотного проектування (потенційні функціональні можливості як сервіси з розуміння застарілої системи). Метод SMART зосереджується на розробці цільової системи на основі потенційних функціональних можливостей як послуг і оцінці їх із зацікавленими сторонами, беручи до уваги різні функціональні та нефункціональні характеристики цільової

системи. З точки зору технічних характеристик, Cuadrado et al. [7] надають чітке пояснення використання специфікації OSGi та сервісної платформи. Автори вважають зручність обслуговування та сумісність важливими критеріями цільової системи та відповідно використовують специфікації OSGi для підтримки цих нефункціональних характеристик.

Одним із важливих етапів з організаційної точки зору є визначення здійсненності еволюції, яка визначає хід або ні хід проекту еволюції. Визначення здійсненності еволюції зосереджується на економічній і технічній оцінці успадкованої системи та цільової системи разом із бізнес-цілями, яких організація хоче досягти шляхом еволюції. На етапі визначення здійсненності еволюції використовуються дані про розуміння успадкованої системи (наприклад, складність коду, показники згуртованості та зв'язку тощо) та висновки про розуміння цільової системи (наприклад, нефункціональні характеристики, вибір технології надання послуг, дизайн оркестровки тощо) для визначення технічної та економічної здійсненності. Найкращі практики на етапі визначення здійсненності еволюції включають аналіз витрат і вигод, запропонований Снідом [24] для проектів реінжинірингу, і він служить гарною відправною точкою. Ця модель CBA широко використовувалася в еволюції SOA [14, 27, 28]. Умар і Зордан [27] розширили модель CBA, включивши витрати на інтеграцію, що полегшує прийняття стратегічних рішень у спадок від еволюції SOA. Метод SMART використовує аналіз варіантів для реінжинірингу (OAR) [24] для визначення так званої точки прийняття рішення щодо здійсненності міграції. На основі методу SMART і схеми прийняття рішень Ерраді та ін. [8], Салама та Елі [23] представляють інструмент прийняття рішень для вибору стратегій модернізації спадщини SOA, який також враховує можливість еволюції.

Ми також визначили кілька можливих покращень нашого дослідження. Одним із удосконалень поточного процесу оцінювання є подвійна перевірка результату оцінювання. У представленій оцінці основні статті були розподілені між п'ятьма дослідниками, а потім оцінені. Як удосконалення ми прагнемо

подвійної перевірки кожного результату оцінювання принаймні одним іншим дослідником. Це, безперечно, зменшить упередженість (тобто суб’єктивна класифікація не проводиться) і призведе до більш точних висновків. У своїй оцінці ми задокументували те, про що повідомлялося в статті. Наприклад, методи «архітектурного відновлення» та «архітектурної реконструкції» можна вважати однаковими, і обидві знову можна вважати такими, що підпадають під заголовок «зворотного проектування». У нашій оцінці ми не використовували такі суб'єктивні класифікації. У майбутньому ми прагнемо вдосконалити результати нашої оцінки за допомогою узагальнення атрибутів [71] – способу узагальнити значення знахідки в загальну та пов’язану категорію. Крім того, ми також прагнемо оцінити запропоновану систему оцінювання за допомогою тематичних досліджень і вдосконалити її відповідно. Наразі наше дослідження зосереджено лише на спадщині еволюції SOA, про яку повідомляють наукові кола. У майбутньому ми також прагнемо надати подібну інформацію про підходи до еволюції SOA, які практикуються в промисловості.

3.4 Висновки по розділу

Огляд 121 публікації показав зростаючу увагу до еволюції застарілих систем у контексті SOA, з акцентом на технічні методи, такі як зворотне проектування та обгортання. Хоча дослідження в основному зосереджені на технічному аспекті, відзначено потребу в глибшому розгляді бізнес-факторів та валідації методів у промисловому контексті.

ВИСНОВКИ

Дослідження, проведене в рамках цієї роботи, підкреслює важливість стратегій інтеграції застарілих компонентів у контексті модернізації програмних систем для створення цифрового контенту в сучасну епоху. Аналіз контексту та вимог до створення цифрового контенту показав, що застарілі системи (legacy systems) залишаються значним викликом через їх обмежену сумісність із сучасними технологіями, високі витрати на підтримку та складність інтеграції. Огляд проблеми модернізації виявив необхідність стратегічного підходу до збереження функціональності legacy-компонентів при переході до нових архітектур, тоді як дослідницький підхід підкреслив важливість систематичного аналізу для вибору оптимальних стратегій інтеграції.

Другий розділ, присвячений інженерним методам модернізації, продемонстрував різноманітність підходів до повторного використання застарілих компонентів. Огляд підходів до модернізації показав, що методи, такі як рефакторинг, інкапсуляція та реінжиніринг, дозволяють ефективно інтегрувати legacy-компоненти в сучасні системи. Технічні передумови та виклики, такі як несумісність форматів даних і застарілі мови програмування, вимагають ретельного планування та використання адаптерів чи API. Методологія ServiciFi для реінжинірингу виявилася перспективною завдяки своїй здатності трансформувати застарілі системи в сервісно-орієнтовані архітектури, забезпечуючи модульність і масштабованість. Оцінка ефективності методів модернізації підтвердила, що комбінація кількох стратегій, залежно від специфіки системи, є найбільш результативною.

Третій розділ, що аналізує вплив еволюції сервісно-орієнтованої архітектури (SOA) на модернізацію, підкреслив її ключову роль у полегшенні інтеграції застарілих компонентів. Передумови модернізації через SOA вказують на її здатність забезпечувати гнучкість і повторне використання компонентів через стандартизовані інтерфейси. Метод дослідження та структура оцінки еволюції від legacy-систем до SOA дозволили систематизувати підходи до

трансформації. Огляд первинних досліджень і оцінка ефективності показали, що SOA сприяє зниженню витрат на інтеграцію та підвищенню адаптивності систем, хоча потребує значних початкових інвестицій у реінжиніринг.

Узагальнюючи, стратегії інтеграції застарілих компонентів є критично важливими для забезпечення сумісності legacy-систем із сучасними вимогами цифрового контенту. Результати дослідження можуть бути використані розробниками, архітекторами програмного забезпечення та менеджерами проєктів для планування й реалізації модернізації. Перспективи подальших досліджень включають розробку автоматизованих інструментів для аналізу застарілих систем, вдосконалення гібридних стратегій інтеграції та дослідження впливу хмарних технологій на реінжиніринг legacy-компонентів. Ці напрямки сприятимуть створенню більш ефективних і стійких рішень для модернізації програмних систем у майбутньому.

СПИСОК ПОСИЛАНЬ НА ДЖЕРЕЛА

1. Abowd, G., Goel, A., Jerding, D. F., McCracken, M., Moore, M., Murdock, J. W., Potts, C., Rugaber, S., & Wills, L. MORALE: Mission Oriented Architectural Legacy Evolution. *International Conference on Software Maintenance*, Bari, Italy, October 1997, 150–159. - Режим доступу: https://ieeexplore.ieee.org/document/624307

2. Abril, R. M. European Interoperability Reference Architecture (EIRA) and eGovERA Suite for Public Administration. *KU Leuven Public Governance Institute*, 2023. — Режим доступу: https://www.kuleuven.be/

3. AltexSoft. Legacy System Modernization: How to Transform the Enterprise. *AltexSoft Blog*, 2019. — Режим доступу: https://www.altexsoft.com/blog/legacy-system-modernization

4. Azilen Technologies. Legacy System Integration: Strategies for Seamless Integration. *Medium*, 2024. — Режим доступу: https://medium.com/@azilen/legacy-system-integration-4d7e8f2b3c4a

5. Bisbal, J., Lawless, D., Wu, B., & Grimson, J. Legacy Information Systems: Issues and Directions. *IEEE Software*, 1999, 16(5), 103–111. — Режим доступу: https://doi.org/10.1109/52.795108

6. Brown, A., & Wallnau, K. Framework for Evaluating Software Technology. *IEEE Software*, 1996, 13(5), 39–49. — Режим доступу: https://doi.org/10.1109/52.536457

7. Cha, J.-E., et al. Reengineering Process for Componentization of Legacy System. *Journal of the Korea Society of System Integration*, 2003, 2(1), 111–122.

8. Comella-Dorda, S., Lewis, G., Place, P., Plakosh, D., & Seacord, R. Legacy System Modernization Strategies. *Carnegie Mellon University, Software Engineering Institute*, 2001. — Режим доступу: https://insights.sei.cmu.edu/library/legacy-system-modernization-strategies/

9. Confluent. Integrating Legacy Systems: Challenges and Best Practices. *Confluent Blog*, 2024. — Режим доступу: https://www.confluent.io/blog/integrating-legacy-systems

10. CTI. Bridging the Gap: Strategies for Integrating Legacy Systems with Modern Broadcasting Solutions. *CTI Blog*, 2025. — Режим доступу: https://www.cti.com/blog/legacy-system-integration

11. DevSquad. How to Integrate Legacy Systems: Top Challenges and Strategies. *DevSquad Blog*, 2024. - Режим доступу: https://devsquad.com/blog/legacy-system-integration

12. EDUCAUSE. New Life for Legacy Systems. *EDUCAUSE Review*, 2019. — Режим доступу: https://er.educause.edu/articles/2019/8/new-life-for-legacy-systems

13. GOV.UK. Managing Legacy Technology: Principles for Government. *GOV.UK*, 2019. — Режим доступу: https://www.gov.uk/guidance/managing-legacy-technology

14. GovNet. What Is a Legacy System? Challenges and Modernization Strategies. *GovNet Blog*, 2024. — Режим доступу: https://blog.govnet.co.uk/what-is-a-legacy-system

15. Hein, A. M. Legacy Systems in Space Exploration: Reuse and Validation. *Technical University of Munich*, 2018. - Режим доступу: https://www.tum.de/

16. Hyland. Understanding Legacy Systems: Modernization Strategies. *Hyland Blog*, 2024. - Режим доступу: https://www.hyland.com/en/resources/articles/legacy-systems

17. Impact. Legacy Systems in Digital Transformation: Risks & Challenges. *Impact Blog*, 2024. — Режим доступу: https://www.impactmybiz.com/legacy-systems-digital-transformation

18. Kazman, R., Woods, S. G., & Carriere, S. J. Requirements for Integrating Software Architecture and Reengineering Models: CORUM II. *Fifth Working Conference on Reverse Engineering*, Honolulu, Hawaii, October 1998, 154–163. — Режим доступу: https://ieeexplore.ieee.org/document/732706

19. Kim, H. K., & Chung, Y. K. Transforming a Legacy System into Components. *Computational Science and Its Applications – ICCSA 2006*, 2006, 3982, 198–207. — Режим доступу: https://doi.org/10.1007/11751595_22

20. https://link.springer.com/chapter/10.1007/978-3-540-75912-6_17?utm_source=chatgpt.com

21. LK Technologies. Legacy Systems: Key Insights and Challenges. *LK Tech Blog*, 2024. - Режим доступу: https://lktechnologies.com/legacy-systems-insights

22. https://link.springer.com/chapter/10.1007/978-3-642-16132-2_42?utm_source=chatgpt.com

23. NIST. Supporting Digital Transformation with Legacy Components. *NIST Blog*, 2021. — Режим доступу: https://www.nist.gov/blogs/supporting-digital-transformation-legacy-components

24. Omar, A., & Weerakkody, V. The Impact of Legacy Systems on Digital Transformation in European Public Administration. *ScienceDirect*, 2023. — Режим доступу: https://www.sciencedirect.com/science/article/pii/S0740624X23000883

25. OpenLegacy. A Deep Dive into Legacy System Integration. *OpenLegacy Blog*, 2023. — Режим доступу: https://www.openlegacy.com/blog/legacy-system-integration

26. OpenLegacy. Navigating the Risks of Legacy System Integration. *OpenLegacy Blog*, 2023. - Режим доступу: https://www.openlegacy.com/blog/risks-legacy-system-integration

27. SnapLogic. Legacy System Integration: Explanation & Overview. *SnapLogic Blog*, 2024.-Режим доступу: https://www.snaplogic.com/glossary/legacy-system-integration

28. Talend. What is a Legacy System? Modernization Strategies. *Talend Blog*, 2019. - Режим доступу: https://www.talend.com/resources/what-is-legacy-system

29. Ulrich, W. *Legacy Systems: Transformation Strategies.* — Englewood Cliffs, NJ: Prentice Hall, 2002. — 448 с.

30. Workato. How to Integrate Legacy Systems with Modern SaaSApplications. *Workato Blog*, 2024. — Режим доступу: https://www.workato.com /the-connector/legacy-system-integration

БІБЛІОГРАФІЧНА ДОВІДКА

Тема бакалаврської роботи: " Стратегії інтеграції застарілих компонентів"

Обсяг пояснювальної записки: 94 аркушів

Дата закінчення роботи 10 червня 2025р.

Підпис студента _____