https://doi.org/10.31891/2307-5732-2026-365-35
УДК 004.4

ЛЯШКЕВИЧ МАРІЯ
Львівський національний університет імені Івана Франка
https://orcid.org/0000-0002-9655-036X
e-mail: mariia.liashkevych@lnu.edu.ua
ШУВАР РОМАН
Львівський національний університет імені Івана Франка
https://orcid.org/0000-0001-6768-4695
e-mail: roman.shuvar@lnu.edu.ua

РОЗРОБКА АНАЛІТИЧНИХ ІНСТРУМЕНТІВ ДЛЯ ОЦІНКИ ФУНКЦІОНАЛЬНОГО СТАНУ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ПРОТЯГОМ ЖИТТЄВОГО ЦИКЛУ

В роботі подано комплексний підхід до розробки аналітичних інструментів для оцінки функціональних станів програмного забезпечення (ФСПЗ) протягом усього його життєвого циклу (ЖЦ). На відміну від традиційного, ізольованого контролю якості, представлена тут концепція розглядає програмну систему як динамічний об'єкт, що існує в серії формалізованих станів та їх переходів. Функціональний стан, як інтегрована модель, охоплює дані з різних фаз, включаючи збір вимог, архітектурне проектування, впровадження, тестування, розгортання, експлуатацію та обслуговування. Аналітичні засоби забезпечують шість типів аналітики: описовий, діагностичний, прогнозний, прескриптивний, порівняльний та контекстно-адаптивний. Кожний тип аналітики визначає відповідний набір інструментів – моделей, методів, алгоритмів та метрик – для виявлення відхилень, аналізу причинно-наслідкових зв'язків, прогнозування деградації, вибору стратегій реагування та адаптації рішень до операційного середовища. Крім того, пропонується інтегрована система метрик ФСПЗ для оцінки стабільності, надійності, безпеки, адаптивності та ризику програмного забезпечення (ПЗ). Архітектуру аналітичних інструментів подано на основі фреймворку С4, детально описуючи контекстний рівень, контейнерний рівень, компонентний рівень та рівень коду. Також наведено структурні діаграми та діаграми поведінки, формати даних, інструменти візуалізації та приклади інтерфейсу користувача. Крім того, у статті досліджуються сценарії застосування аналітики ФСПЗ у тестовому режимі. В сукупності, запропоновані засоби дозволяють встановити керований, адаптивний та саморегульований шлях розробки ПЗ, що є критично важливим для складних, високоризикованих та динамічних програмних середовищ. Результати можуть бути використані для розробки систем підтримки рішень для розробки, тестування, розгортання та обслуговування ПЗ.

Ключові слова: *програмне забезпечення, прийняття рішень, інформаційна технологія, аналітика, функціональний стан програмного забезпечення.*

LYASHKEVYCH MARIIA, SHUVAR ROMAN
Ivan Franko National University of Lviv

DEVELOPMENT OF ANALYTICAL TOOLS FOR SOFTWARE FUNCTIONAL STATE ESTIMATION THROUGHOUT THE LIFECYCLE

This paper presents a comprehensive approach to developing analytical tools for assessing software functional states (SFS) throughout the software development life cycle (SDLC). In contrast to traditional, isolated quality control, the concept presented here considers a software system as a dynamic object that exists in a series of formalized states and their transitions. The functional state, as an integrated model, encompasses data from different phases, including requirements gathering, architectural design, implementation, testing, deployment, operation, and maintenance. Analytics tools provide six types of analytics: descriptive, diagnostic, predictive, prescriptive, comparative, and contextually adaptive. Each type of analytics defines a corresponding set of tools – models, methods, algorithms and metrics – for detecting deviations, analyzing cause-and-effect relationships, predicting degradation, choosing response strategies and adapting solutions to the operating environment. In addition, an integrated system of SFS metrics is proposed for assessing the stability, reliability, security, adaptability and risk of software. The architecture of the analytical tools is presented based on the C4 framework, describing in detail the context level, container level, component level and code level. Also, the structural and behavioral diagrams are provided, data formats, visualization tools, and user interface examples. In addition, the article explores scenarios for the application of SFS analytics in test mode. Taken together, the proposed tools allow for a controlled, adaptive, and self-regulating path for software development, which is a critical point for complex, high-risk, and dynamic software environments. The results can be used to develop decision support systems for software development, testing, deployment, and maintenance.

Keywords: *software, decision making, information technology, analytics, software functional state.*

Стаття надійшла до редакції / Received 13.02.2026
Прийнята до друку / Accepted 11.04.2026
Опубліковано / Published 28.05.2026

Постановка проблеми у загальному вигляді та її зв'язок із важливими науковими чи практичними завданнями

Сучасна розробка програмного забезпечення характеризується різноманітністю методологій розробки, які визначають структуру ЖЦ, ролі зацікавлених сторін, артефакти, механізми забезпечення якості та управління ризиками та ін. Різні підходи, від каскадної моделі до гнучкої та спіральної розробки, і навіть їх комбінації, суттєво впливають на якість, безпеку, терміни поставки та вартість ПЗ. Вибір конкретної методології залежить від стабільності вимог, критичності області застосування, рівня ризику, зрілості команди та використання компонентів штучного інтелекту (ШІ). Проте, незалежно від використаної методології, послідовність фаз життєвого циклу програмного забезпечення (ЖЦПЗ) дотримується універсальної логіки, що охоплює ініціювання та планування, аналіз вимог, проектування архітектури, організацію команди, розробку, тестування,

розгортання та підтримку. Кожна фаза, зазвичай, передбачає виконання кількох етапів та генерує велику кількість рішень, подій, артефактів та метрик, які разом формують складну, багатовимірну картину стану ПЗ. Зі зростанням розміру та динаміки проектів стає практично неможливо належним чином обробити ці всі дані без спеціалізованих аналітичних інструментів. Важливість розробки інструментів для аналітики ФСПЗ випливає з кількох фундаментальних факторів наведених нижче.

По-перше, традиційні метрики, такі як кількість дефектів, покриття тестами, складність коду, продуктивність та окремі метрики безпеки описують властивості ізольованого ПЗ. Хоча вони можуть виявити наявність проблем, вони не відображають загальний ФСПЗ та його еволюцію з часом. Одна й та сама метрика може інтерпретуватися по-різному на різних етапах ЖЦПЗ, у різних середовищах виконання або в різних архітектурних контекстах.

По-друге, сучасне ПЗ характеризуються зростаючою складністю та стохастичністю. Мікросервісна архітектура, хмарні сервіси, інтеграція із зовнішніми API, процеси безперервної інтеграції/безперервної доставки (CI/CD) та широке використання компонентів, керованих даними, та управління життєвим циклом (УЖЦ), разом формують нелінійну, контекстно-залежну поведінку ПЗ. У цьому контексті детерміновані припущення щодо стабільності атрибутів більше не є доречними, а оцінка ПЗ виключно на основі статичних або усереднених показників більше не є достатньою.

По-третє, у сучасному середовищі DevSecOps та ШІ раннє прогнозування ризиків є критично важливим. Найсерйозніші проблеми, втрата якості, технічний борг чи вразливості безпеки, розвиваються поступово, починаючи з ранніх архітектурних рішень або рішень щодо вимог. Традиційні методи можуть виявити їх лише після того, як ризики вже матеріалізувалися, тоді як аналіз на основі станів може виявити ледь помітні ознаки та потенційно небезпечні зміни в проектуванні системи до того, як відбудуться збої.

У цьому контексті, аналіз функціонального стану ФСПЗ формує основу для інтелектуального управління ЖЦПЗ. ПЗ більше не розглядається як ізольовані, окремі артефакти (код, тести, конфігурації чи події), а як набір ФС та переходів між ними. ФСПЗ являють собою інтегровану модель, яка фіксує дані з різних етапів ЖЦ та підтримує скоординоване прийняття рішень, враховуючи довгостроковий вплив цих рішень на стабільність, безпеку та масштабованість ПЗ.

Аналіз досліджень та публікацій

Аналітика ПЗ, за останні роки, перетворилася на галузь, яка перетворює дані з усього ЖЦПЗ на практичні висновки для підтримки рішень щодо розробки та управління ПЗ. Прикладами таких даних можуть бути: вихідний код, безперервна інтеграція/безперервна доставка, помилки, заявки, журнали, дані телеметрії, вимірювання під час виконання, залежності, результати тестування та ін. Дослідження показують, що ця галузь охоплює як традиційний аналіз репозиторіїв ПЗ, так і ширший спектр платформ, які інтегрують артефакти процесів, продуктів та операційної діяльності у цілі “Аналітичні засоби для розробки” [1-2]. Важливою сферою роботи є розгляд аналітики ПЗ як практичної дисципліни, що підтримує команди типу “аналіз → дії”, та зосередження на застосовності метрик та моделей у реальних середовищах розробки [1, 3].

Аналіз коду є досить складним і об’ємним процесом, проте бібліотеки коду та змін, наприклад: коміти, відмінності, власність та журнали змін, є фундаментальними для аналізу еволюції, дефектів та деградації архітектури. Такі дослідження уже ведуться давно. Класичне дослідження [4] демонструє, що складність змін коду може передбачати дефекти, підкреслюючи важливість природи даних, заснованої на змінах.

Відомі методології як “Видобування програмного репозиторію” (MSR - mining software repository), передбачають системну роботу із репозиторієм ПЗ, наприклад: вибір репозиторію, очищення, метрики та валідність [5-6]. Хоча ранні дослідження у [6] були розглянуті питання контексту, з 2007 року ситуація кардинально змінилася, в основному зосереджувалися на видобутку історій версій та артефактів репозиторіїв, сучасні програмні екосистеми вимагають контекстно-орієнтованої, орієнтованої на стан аналітики, яка інтегрує сигнали розробки, архітектури, середовища виконання та керовані ШІ сигнали в уніфіковані моделі функціональних станів. Тому, контекст, який здебільшого обмежувався версіями коду та історією змін у [6], сьогодні став багатовимірним і динамічним, а сучасні огляди узагальнюють MSR-підходи саме для архітектури, наприклад: відновлення архітектури, виявлення ерозії, залежності [7].

Ключовою сферою сучасного аналізу ПЗ є аналіз впливу вимог та проектної документації на якість, ремонтопридатність та безпеку системи. З аналізу літературних джерел відомо те, що характеристики, структура, формалізація та класифікація вимог безпосередньо впливають на ремонтопридатність, безпеку та подальшу еволюцію ПЗ. Зокрема, дослідження автоматизованої класифікації вимог з урахуванням ремонтопридатності як компонента безпеки показують, що ранній аналіз текстових артефактів може виявити потенційні технічні недоліки та джерела ризику на етапі специфікації [8]. Тому аналіз вимог розглядається не лише як інструмент для покращення документації, але й як проактивний підхід до управління якістю та безпекою протягом усього ЖЦПЗ.

Ще одним важливим джерелом аналітичних даних є операційні дані: журнали, системні метрики, дані трасування та звіти про події. Ці дані формують основу методів спостережуваності та аналітики під час виконання. Систематичний огляд виявлення аномалій журналів у сфері глибокого навчання показує, що методи виявлення аномалій журналів стали одним з основних інструментів для діагностичної та прогнозної аналітики, особливо в хмарних та мікросервісних архітектурах [9]. У кількох дослідженнях підкреслюється, що якість попередньої обробки логів, зокрема методів аналізу та структурування логів, суттєво впливає на точність моделі та здатність системи виявляти нові або невідомі відхилення [10]. Це свідчить про те, що підготовку даних під час виконання слід вважати невід’ємною частиною процесу аналізу.

Окремий, більш формалізований клас аналітики виконання представлений підходами верифікації під час виконання (ВВ). На відміну від статистичних методів або методів машинного навчання (МН), ВВ спирається на формальні специфікації властивостей системи та контролює їх виконання під час виконання ПЗ. Огляд поточних проблем у цій галузі демонструє, що верифікація під час виконання є перспективним інструментом для забезпечення коректності, безпеки та відповідності системи формальним вимогам у складних та критичних доменах [11]. Додатково досліджується проблема невизначеності та неповноти інформації під час виконання, що особливо актуально для адаптивних та стохастичних систем [12]. Таким чином, верифікація під час виконання формує "жорсткий" рівень аналітики виконання, який можна інтегрувати з більш гнучкими методами спостереження та МН. Разом, домени аналітики вимог, аналізу операційних даних та формальної перевірки під час виконання утворюють багаторівневу структуру для комплексної оцінки ФСПЗ протягом усього його ЖЦ.

Формальний опис ФСПЗ здійснює якісний стрибок у ЖЦПЗ, розширюючи аналітичні можливості [13]. На відміну від аналізу ізольованого аналізу окремих метрик (дефектів, продуктивності, покриття тестами, інцидентів та ін.), ФСПЗ інтегрує різнорідні дані в єдиний стан та його модель переходу. Це дозволяє нам перейти від фрагментованого моніторингу до систематичного, орієнтованого на стан аналізу, розглядаючи ПЗ як динамічний об'єкт із визначеною еволюційною траєкторією.

За наявності ФСПЗ стає можливим реалізувати повний цикл аналітики:

- описову — через узгоджену інтерпретацію показників у межах конкретного стану;
- діагностичну — через аналіз причин переходів між станами;
- прогнозну — через моделювання ймовірностей деградації або вразливості;
- прескриптивну — через автоматизований вибір стратегій реагування;
- порівняльну/стратегічну — через оцінювання траєкторій розвитку різних систем або релізів;
- контекстно-адаптивну — через урахування фази ЖЦ, середовища виконання та доменних ризиків.

Таким чином, ФСПЗ виступає інтеграційною аналітичною основою, що поєднує дані з вимог, архітектури, реалізації, тестування та експлуатації в єдину систему показників. Це забезпечує можливість не лише виявляти поточні проблеми, але й прогнозувати майбутні ризики, безпеку, оцінювати альтернативні сценарії розвитку та формувати обґрунтовані управлінські рішення.

У стратегічному вимірі наявність ФСПЗ трансформує ЖЦ ПЗ з послідовного набору процесів у керований, адаптивний та саморегульований контур, що особливо актуально для DevSecOps-, ШІ-орієнтованих та високоризикових систем. Отже, формалізований ФСПЗ є необхідною передумовою для побудови повноцінної інтелектуальної аналітики ПЗ та систем підтримки прийняття рішень у сучасній інженерії ПЗ.

Описовий аналіз має на меті формально відобразити та пояснити поточний стан ПЗ, процес його розробки та середовище його роботи, щоб охопити фактичні характеристики, тенденції та закономірності його функціонування. Типові інструменти для описової аналітики: dashboards, агреговані метрики якості, тренди дефектів, змін чи покриття, профілі компонентів, "карти" залежностей. Узагальнені огляди аналітики ПЗ підкреслюють, що описова аналітика є найпоширенішою, але сама по собі часто не дає керованих рішень без зв'язку з контекстом [1-2].

Мета діагностичного аналізу полягає у визначенні причин погіршення якості, дефектів або інцидентів шляхом розпізнавання причинно-наслідкових зв'язків між подіями, змінами та функціями в програмній системі. В галузі MSR були запропоновані методи аналізу історії змін, оцінки власності компонентів, ідентифікації так званих файлів "гарячих точок", кластеризації джерел дефектів та виявлення закономірностей еволюції програмного забезпечення [5-6]. На рівні виконання діагностична аналітика базується на аналізі журналів подій та даних телеметрії. У цьому огляді методів глибокого навчання для виявлення аномалій журналів систематично окреслено архітектури моделей, методи представлення та структурування журналів, а також вимоги до попередньої обробки для забезпечення точності та відтворюваності результатів аналізу [9].

Прогнозна аналітика застосовується для пошуку обґрунтованих прогнозів щодо майбутнього ФСПЗ, за умови, що поточні тенденції розвитку залишаються незмінними. Її основні цілі полягають у прогнозуванні дефектів компонентів, ризиків деградації архітектури, зростання технічного боргу та ймовірності переходу системи у вразливий або критичний стан. Окрім класичних однопроектних та міжпроектних моделей [14], сучасні дослідження також зосереджені на питаннях зсуву набору даних та дрейфу ознак у довгострокових проектах. У роботі [3] продемонстрували вплив часового дрейфу на стабільність моделей МН та запропонували адаптивну стратегію перенавчання. Також постійно розвиваються методи прогнозної аналітики, засновані на великих мовних моделях (LLM) [15] та інтеграції з runtime-даними, даними середовища виконання. Ці моделі поєднують телеметричні дані часових рядів виконання (журнали, метрики, трасування) зі змінами коду та процесів, що дозволяє раннє виявлення аномалій та прогнозування подій у середовищах DevOps. Збір та аналіз телеметричних даних середовища виконання підвищує обізнаність про стан системи та виявляє потенційні проблеми, дозволяючи прогнозувати до того, як у робочому середовищі відбудуться фактичні збої [16]. Крім того, у [17] представили узагальнюючий огляд пояснювального ШІ (XAI) для передбачення дефектів ПЗ, наголошуючи на необхідності інтерпретованості моделей для їх прийнятності в DevOps-процесах. Це безпосередньо пов'язано з переходом від суто прогнозної до прескриптивної аналітики.

Прескриптивна аналітика спрямована на формування обґрунтованих рекомендацій щодо подальших дій. В сфері розроблення ПЗ це є рефакторинг, оптимізація тестової стратегії, вибір політики релізів або заходів з мінімізації ризиків. У науковій літературі перехід від описової та прогнозної аналітики до прескриптивної часто

пов'язується з управлінням технічним боргом, який може бути кількісно виміряний, пріоритезований та інтегрований у процес прийняття рішень [18-19]. Окремі дослідження аналізують практичні метрики та обмеження вимірювання технічного боргу в реальних проєктах [20]. Для задач продуктивності та надійності описано підходи адаптації аналітичних моделей до індустріальних обмежень даних, архітектурної складності та експлуатаційного контексту [21].

Порівняльна аналітика ПЗ спрямована на оцінку команд, ПЗ і процесів з метою формування довгострокових управлінських рішень. У наукових дослідженнях значна увага приділяється метрикам продуктивності розробки, зокрема підходам DevOps та DORA, які пов'язують частоту релізів, рівень відмов із організаційною ефективністю. Крім процесних метрик, стратегічний аналіз охоплює архітектурні аспекти, еволюцію систем та якість рішень, що відображено у систематичних дослідженнях MSR для програмної архітектури [7]. Таким чином, стратегічна аналітика інтегрує технічні та організаційні показники, забезпечуючи основу для обґрунтованого вибору методологій, архітектурних стилів і політик управління якістю.

Контекстно-адаптивна аналітика спрямована на врахування фази ЖЦ, середовища виконання, архітектурної конфігурації та невизначеності даних під час оцінки ПЗ. Вона поєднує верифікацію в режимі реального часу, що забезпечує моніторинг виконання відносно формальних специфікацій [11], з підходами "log-based anomaly detection" на основі глибокого навчання [9]. Дослідження також показують суттєвий вплив підготовки логів на якість моделей [10]. Таким чином, аналітика переходить від статичних метрик до динамічного, стано-орієнтованого контролю поведінки ПЗ.

Водночас, в існуючій науковій літературі бракує способу інтеграції цих методів в єдину модель, орієнтовану на ФСПЗ, яка може узгодити дані з різних етапів ЖЦПЗ. Формалізація ФС має вирішальне значення для забезпечення переходу до цілісного, контекстно-адаптивного аналізу [13].

Формулювання цілей статті

Метою роботи є: розробка аналітичних інструментів для оцінки ФСПЗ протягом усього ЖЦ на основі багаторівневої моделі аналітики, що поєднує описові, діагностичні, прогнозні, прескриптивні, порівняльно-стратегічні та контекстно-адаптивні механізми аналізу. Запропонований підхід спрямований на переосмислення ЖЦПЗ як керованої, адаптивної та саморегульованої системи, у межах якої управління здійснюється на основі ФС і їх динаміки, а не лише через послідовність ізольованих процесів.

У межах дослідження визначається мінімально достатній набір аналітичних інструментів, необхідних для реалізації зазначеної моделі, при цьому основна увага зосереджується на методологічних і архітектурних аспектах проєктування відповідного ПЗ.

Виклад основного матеріалу

Засоби аналітики ФСПЗ є складними для імплементації не лише через технічну багатовимірність, а й через концептуальну неоднорідність середовища, у якому вони застосовуються.

По-перше, вони працюють із великою кількістю різнорідних джерел даних: вихідна розмова про вимоги, можливий код, історії змін, вимоги, тести, інциденти, журнали виконання, телеметрія, метрики продуктивності, дані безпеки, онтології тощо. Ці дані мають різні формати (CSV, JSON, логи, OWL, API-відповіді), різну структуру, часову природу (статичні та потокові) і різний рівень семантичної формалізації. Тому необхідно передбачити механізми інтеграції, нормалізації, синхронізації та узгодження семантики.

По-друге, різні типи аналітики (описова, діагностична, прогнозна, прескриптивна, порівняльна та контекстно-адаптивна) спираються на різні алгоритми, моделі та припущення. Методи МН, марковські моделі, Монте Карло-симуляції, генетичні алгоритми (ГА), підходи XAI чи формальні методи верифікації потребують різних атрибутів даних і різних схем підготовки ознак. Це ускладнює побудову універсальної архітектури, яка б залишалася модульною та розширюваною.

По-третє, складність має методологічний вимір: необхідно враховувати різні сценарії застосування, фази SDLC, рівні критичності системи, невизначеність і змінність вимог. Узгодження моделей, метрик і рішень у довгостроковій перспективі вимагає високої гнучкості та конфігурованості, що є критично важливим для практичної ефективності аналітичної системи.

По-четверте, на сьогодні, відсутнє єдине уніфіковане представлення знань про програмний продукт, тому, дані зберігаються у різних структурах залежно від підсистеми. Наприклад, вимоги, за звичай, у JSON-моделі після LLM-обробки, аналітичні результати зберігаються у табличних наборах або time-series, онтологія у OWL-об'єктах, а результати оптимізації у власних структурах Монте Карло та ГА. Така фрагментація призводить до низки проблем під час передачі даних між компонентами сценаріїв, а саме:

- виникає необхідність постійних трансформацій форматів, що ускладнює код і підвищує ризик втрати семантики.
- відсутність єдиної онтологічної або метамоделі означає, що одна й та сама сутність, наприклад, вимога або ризик, може інтерпретуватися по-різному в різних модулях.
- сценарії, які комбінують кілька типів аналітики, змушені покладатися на неформальні припущення щодо структури даних, що знижує узгодженість і масштабованість системи.

У результаті система функціонує як набір взаємодіючих, але семантично частково ізольованих компонентів. Це обмежує можливість побудови повністю інтегрованого, стано-орієнтованого контуру управління та ускладнює подальше розширення архітектури. У зв'язку з цим, за основу взято модель ФСПЗ, яку описано в [13], а також запропонований фреймворк у [22]. Опис програмного забезпечення зроблено у відповідності до архітектурного фреймворку C4 та засобу проектування UML - PlantUML.

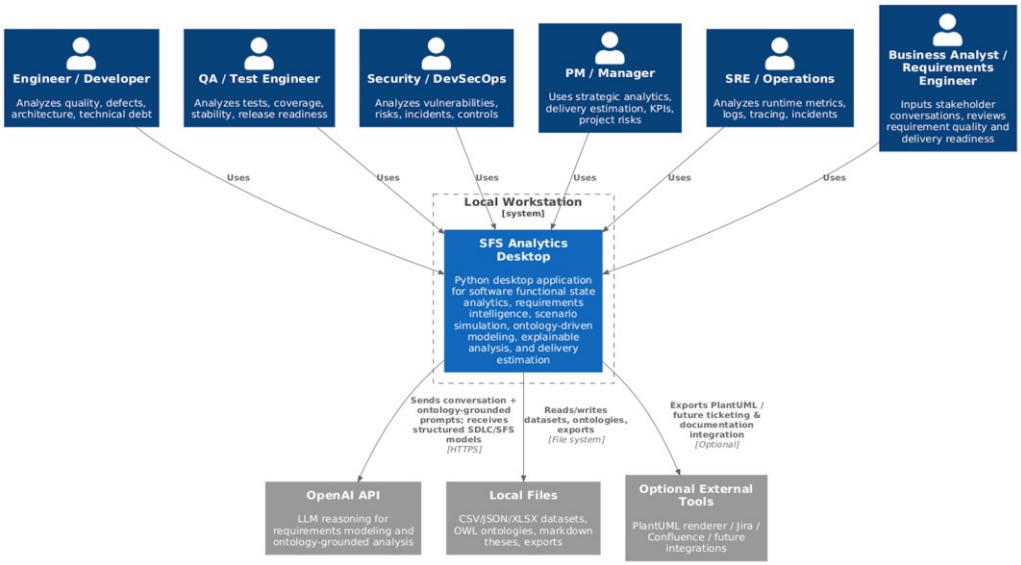


Рис. 1. Високо-рівневий контекст системи

Діаграма на рис. 1 відображає “Аналітика ФСПЗ для робочого столу” (SFS Analytics Desktop) як центральну аналітичну систему, що працює локально на одній робочій станції та взаємодіє з різними ролями користувачів і зовнішніми сервісами. Користувачами виступають інженер/розробник, спеціаліст по забезпеченню якості (QA), спеціаліст із безпеки (DevSecOps), менеджер проєкту, інженер щодо надійності сайту (SRE - Site Reliability Engineering) та бізнес-аналітик. Кожна роль використовує систему для аналізу функціонального стану ПЗ, ризиків, вимог, якості, інцидентів або для стратегічного планування.

Система інтегрується із зовнішнім OpenAI API для інтелектуального аналізу розмов, формалізації вимог та XAI. Також вона взаємодіє із локальною файловою системою, з якої завантажуються датасети (CSV, JSON), OWL-онтології та інші артефакти. Таким чином, у контексті С4 система виступає як автономний інтелектуальний аналітичний центр, що поєднує локальні дані, зовнішні LLM-сервіси та багаторольову взаємодію для підтримки прийняття рішень протягом усього ЖЦПЗ.

Контейнер-діаграма (рис. 2) деталізує внутрішню структуру ПЗ на рівні основних логічних контейнерів. Система складається з “Desktop UI (Tkinter)”, який забезпечує багатовкладковий інтерфейс: “Framework”, “Analytics Results”, “SFS Dashboard”, “Scenarios”, “Requirements Chat” та “Ontology & Optimization”. “UI” взаємодіє з “Application Core”, що координує виконання сценаріїв, аналітичних методів та сервісів моделювання.

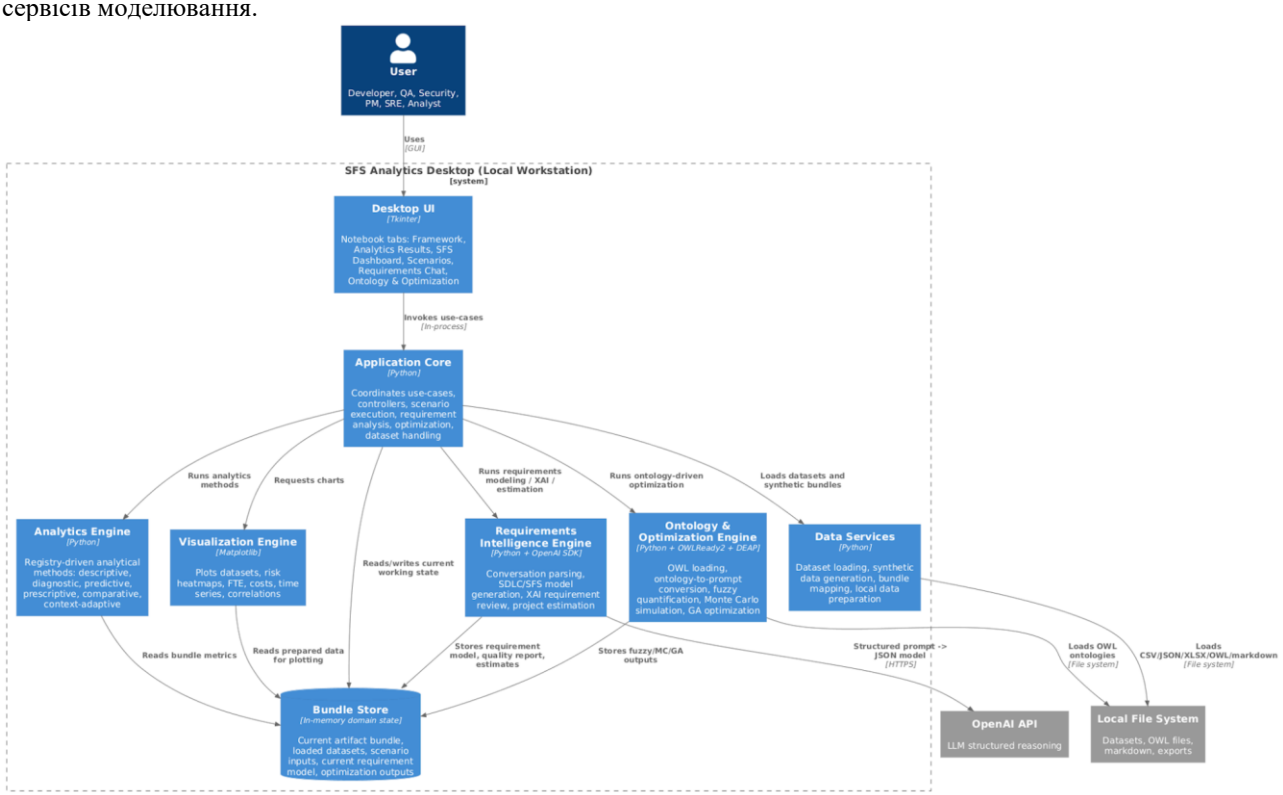


Рис. 2. Подання системи на рівні контейнерів

Рушій аналітики (Analytics Engine) реалізує набір методів описової, діагностичної, прогнозної, прескриптивної, порівняльної та контекстно-адаптивної аналітики через механізм реєстрації. Інтелектуальний рушій вимог (Requirements Intelligence Engine) відповідає за обробку розмов, виклики OpenAI API, аналіз якості вимог, XAI та оцінку вартості проекту у еквіваленті повної зайнятості (FTE estimation). Рушій для онтології та оптимізацій (Ontology & Optimization Engine) забезпечує завантаження OWL-онтологій, генерацію підказок на основі онтології (ontology-grounded prompt), нечітке квантування, Монте Карло симуляцію та оптимізацію за допомогою ГА. Рушій для візуалізації (Visualization Engine) відображає графіки, теплові карти ризиків та оцінки ресурсів. Сервіси даних (Data Services) керують завантаженням датасетів і синтетичних даних. Внутрішнє сховище (Bundle Store) зберігає поточний стан моделі, результати аналітики та оптимізації. Зовнішня взаємодія обмежується OpenAI API та локальною файловою системою.

Діаграма компонентів (Desktop UI) показує внутрішню структуру графічного інтерфейсу. Центральним елементом є головна форма “MainWindow”, яка керує навігацією між вкладками та координує взаємодію з ядром ПЗ (Application Core). Ліва панель (Control Panel) забезпечує вибір персони, фази SDLC, типу аналітики, методу та завантаження синтетичних даних.

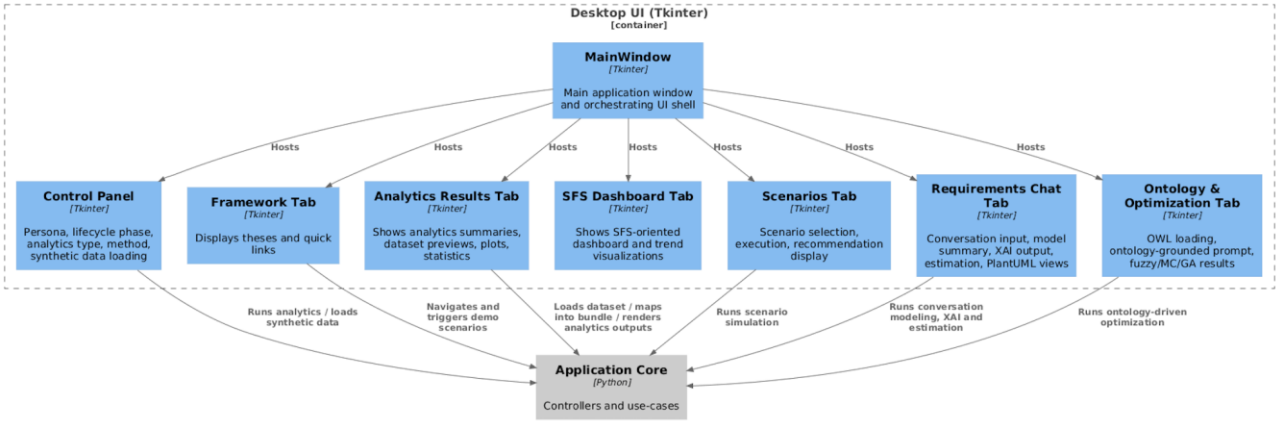


Рис. 3. Діаграма компонентів

Вкладка “Framework” відображає методологічні матеріали та швидкі сценарії (quick links). “Analytics Results” відповідає за відображення результатів аналітичних методів, таблиць та графіків на основі завантажених датасетів. “SFS Dashboard” забезпечує стано-орієнтовану візуалізацію ФСПЗ. “Scenarios” дозволяє запускати багаторівневі аналітичні сценарії та формувати рекомендації. Вкладка “Requirements Chat” реалізує аналіз розмов, структуризацію вимог, XAI, оцінку проєкту та генерацію PlantUML. “Ontology & Optimization” забезпечує завантаження OWL-онтологій, нечітке квантування, Монте Карло та ГА. Усі “UI”-компоненти взаємодіють із “Application Core” через чітко визначені запити, не містячи бізнес-логіки всередині інтерфейсу, що забезпечує модульність та розширюваність архітектури.

Діаграма компонентів ядра системи (Application Core) реалізує координацію аналітики, моделювання вимог та оптимізацію. Центральним компонентом є “Controller”, що забезпечує доступ до реєстру аналітичних методів та виконує їх над поточним набором артефактів (bundle). “Scenario Runner” організовує багаторівневі сценарії, комбінуючи кілька типів аналітики та формуючи рекомендації.

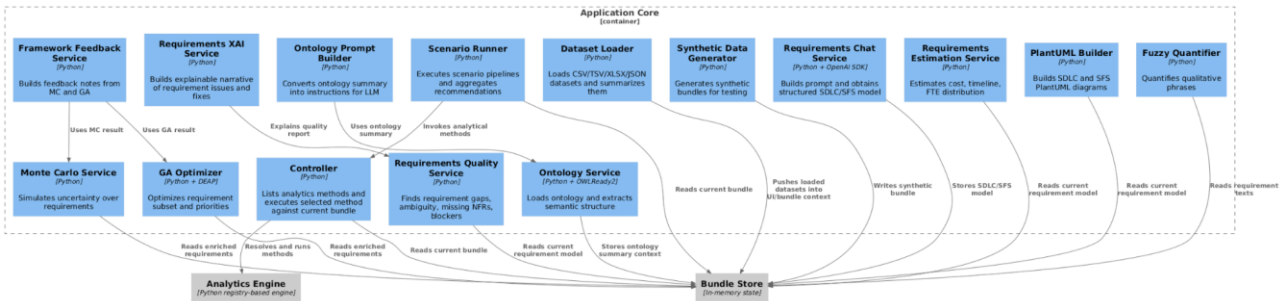
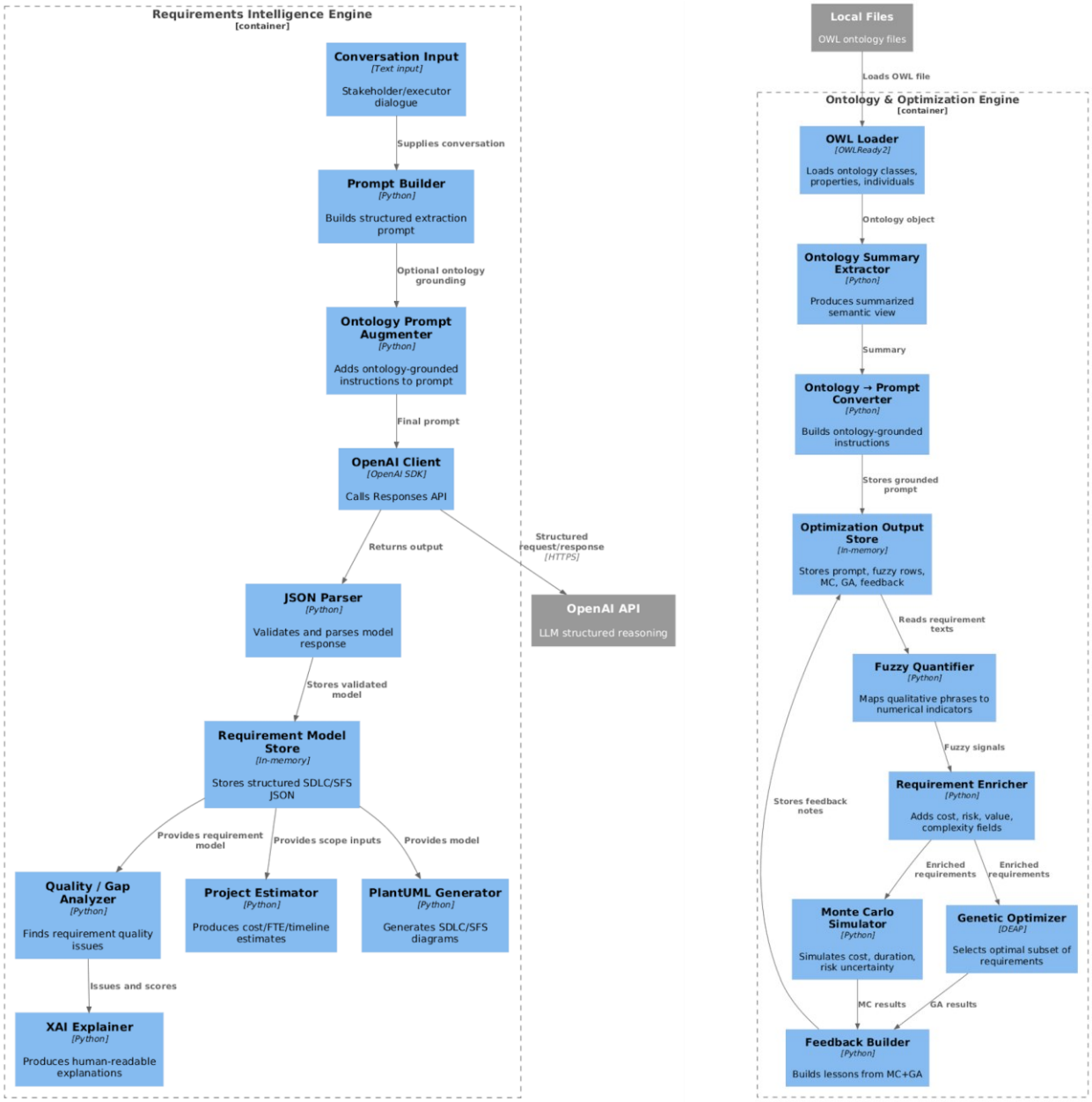


Рис. 4. Діаграма компонентів рушія інтелектуального аналізу вимог

“Dataset Loader” та “Synthetic Data Generator” відповідають за підготовку даних для аналітичних експериментів, що особливо важливо на епатпі тестування розроблюваних засобів аналітики. Підсистема “Requirements Intelligence” включає “Requirements Chat Service” (виклик OpenAI API), “Requirements Quality Service” (аналіз прогалин), “Requirements Estimation Service” (оцінка FTE/вартості) та “Requirements XAI Service” (пояснення результатів). “PlantUML Builder” генерує діаграми ЖЦПЗ та ФСПЗ. Підсистема “Ontology & Optimization” складається з “Ontology Service” (зчитування OWL), “Ontology Prompt Builder”, “Fuzzy Quantifier”, “Monte Carlo Service” та “GA Optimizer”, які формують інтегрований контур оцінки невизначеності

та оптимального набору вимог. Усі компоненти взаємодіють через внутрішнє сховище стану “Bundle Store”, що забезпечує узгодженість моделі та результатів.

Діаграми компонентів розвідки вимог показано на рис. 5.а, для “Ontology & Optimization” на рис. 5.б. Розвідка вимог - підсистема, що перетворює неструктуровану розмову у формалізовану модель ЖЦПЗ/ФСПЗ з подальшим пояснювальним та економічним аналізом. Початковим компонентом є “Conversation Input”, який приймає текст розмови між зацікавленими сторонами про ПЗ. “Prompt Builder” формує структурований запит до LLM, визначаючи очікувану JSON-схему (FR, NFR, constraints, open questions, SDLC phases, SFS transitions). За наявності онтології “Ontology Prompt Augmenter” додає семантичні обмеження.



а) б)

Рис. 5. Діаграми компонентів для: а) розвідки вимог; б) онтології та оптимізації

OpenAI Client здійснює виклик API та отримує структурований результат. JSON Parser перевіряє та нормалізує модель, яка зберігається у “Requirement Model Store”. Далі Quality Analyzer оцінює повноту, неоднозначність та ризики вимог. XAI пояснювач формує пояснювальний звіт із рекомендаціями щодо покращення. OpenAI Client здійснює виклик API та отримує структурований результат. JSON Parser перевіряє та нормалізує модель, яка зберігається у “Requirement Model Store”. Далі Quality Analyzer оцінює повноту, неоднозначність та ризики вимог. XAI Explainer формує пояснювальний звіт із рекомендаціями щодо покращення. У сукупності ця підсистема забезпечує повний контур: “структуризація → аналіз → пояснення → оцінка → візуалізація”.

Компонент “Ontology & Optimization” (рис. 5.б) описує підсистему, що інтегрує семантичне моделювання з аналітикою невизначеності та оптимізацією. Початковим компонентом є “OWL Loader”, який зчитує онтологію (OWL/RDF) та формує внутрішнє представлення класів, властивостей і індивідів. “Ontology

Summary Extractor" узагальнює семантичну структуру, виділяючи доменні поняття (вимоги, ризики, ролі, фази ЖЦПЗ, ФСПЗ). Далі "Ontology → Prompt Converter" формує інструкції для подальшого аналізу або LLM-викликів. "Fuzzy Quantifier" перетворює якісні формулювання (наприклад, "high security", "very fast") у числові показники. "Requirement Enricher" доповнює вимоги атрибутами вартості, ризику та цінності. На основі цього Монте Карло симулятор моделює невизначеність термінів та витрат, а "Genetic Optimizer (DEAP)" шукає оптимальний набір вимог з урахуванням обмежень бюджету та ризику. "Feedback Builder" формує рекомендації, а "Output Store" зберігає результати (prompt, fuzzy-оцінки, MC/GA-висновки) для подальшої візуалізації та прийняття рішень.

Приклад сценарію: "Conversation → OpenAI → Quality Analysis → XAI → Cost/FTE Estimation" наведено у вигляді діаграми послідовності (рис. 6).

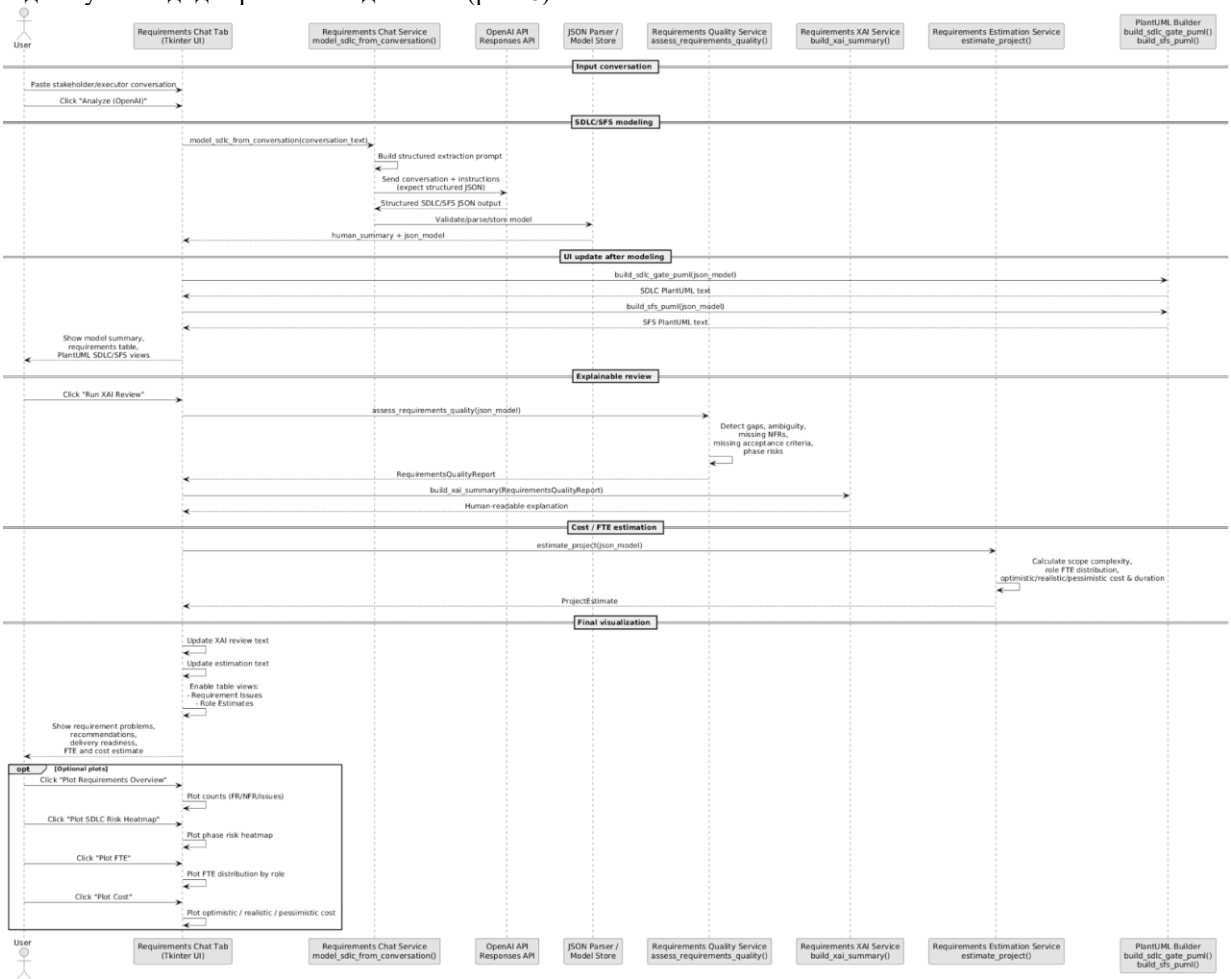


Рис. 6. Приклад сценарію "Conversation → OpenAI → Quality Analysis → XAI → Cost/FTE Estimation"

На першому етапі ми здійснюємо перехід від неструктурованої розмови між зацікавленими сторонами до формалізованої моделі ЖЦПЗ/ФСПЗ. Користувач (аналітик, PM або архітектор) вводить текстову розмову у вкладці "Requirements Chat" та ініціює аналіз. "UI" передає текст до сервісу "model_sdlc_from_conversation()", який формує структурований запит із вимогами до JSON-формату відповіді. У запиті задається схема: функціональні вимоги, нефункціональні вимоги, обмеження, відкриті питання, ЖЦПЗ-фази, гіпотези переходів ФСПЗ. "OpenAI API" виконує когнітивне перетворення тексту розмови у структурований JSON. Після отримання відповіді сервіс валідує її, нормалізує та зберігає у внутрішньому сховищі моделі. Далі будуються два типи PlantUML-діаграм: модель рішення для ЖЦПЗ та модель стану ФСПЗ. Таким чином відбувається формалізація розмови у модель ЖЦ і ФСПЗ. Цей етап трансформує неструктуровану комунікацію у артефакт, який може бути використаний для подальшого аналізу, оптимізації та оцінки ризиків.

На другому етапі користувач запускає XAI аналіз. Сервіс "assess_requirements_quality()" досліджує вимоги на наявність дефектів специфікації: відсутність критеріїв для приймання, нечіткі формулювання, відсутні нефункціональні вимоги, відсутні обмеження, надлишкові відкриті питання. Для кожного виявленого дефекту визначається тип проблеми, її критичність та пов'язана фаза ЖЦПЗ. Далі "build_xai_summary()" формує пояснення природною мовою: чому саме вимога вважається проблемною, які наслідки це матиме на проектування, тестування або реліз, та які конкретні кроки необхідні для покращення. Таким чином формується не просто список помилок, але й проводиться причинно-наслідковий аналіз. На рівні моделі це означає перехід від структурованої інформації до інтерпретованого управлінського висновку. Система оцінює повноту

(completeness score), готовність (readiness score) та впевненість у доставці (delivery confidence). Користувач отримує рекомендації щодо уточнення вимог, додавання метрик, зниження невизначеності та ін.

На останньому третьому етапі моделюється економічна та ресурсна оцінка проєкту. Сервіс “estimate_project()” аналізує кількість та типи вимог, їхню складність і невизначеність, після чого обчислює коефіцієнт складності проєкту. На основі цього розраховуються FTE для різних ролей: PM/BA, архітектор, backend/frontend розробники, LLM-інженер, QA, DevOps, UX, security. Алгоритм генерує три сценарії: optimistic, realistic, pessimistic — із відповідними строками та бюджетами. Таким чином система враховує невизначеність вимог та ризики ЖЦПЗ. Результати відображаються у текстовій формі та у вигляді графіків: розподіл FTE, розподіл витрат, а також потенційна тривалість проєкту. Цей етап перетворює вимоги у прогноз можливість доставки “delivery capability”. Вказана послідовність реалізує повний контур: “Комунікація → Формалізація → Виявлення ризиків → Пояснення → Ресурсне моделювання → Підтримка управлінського рішення”.

Головна форма ПЗ (рис. 7) має ліву панель керування та праву область із вкладками. Активна вкладка “Framework” відображає концептуальні тези фреймворку.

Рис. 7. Приклад основної форми та закладки “Framework”

У верхній частині розташовані “Quick links”, сценарні пресети (Pre-sales, DevSecOps loop, Tech debt) та “розумні” демо-кнопки. Праворуч — запуск вибраного сценарію. Основна область призначена для відображення файлу-розмітки з теоретичною моделлю. Наразі показано повідомлення про відсутність файлу “framework_theses.md”.

Вкладка “Requirements Chat” (рис. 8) призначена для перетворення неструктурованої розмови у формалізовану модель ЖЦПЗ та ФСПЗ. Ліва частина містить текстове поле “Conversation (stakeholders ↔ executers)”, де вводиться діалог між учасниками розмови, тощо. Внизу, кнопки “Load sample” та “Analyze (OpenAI)” для запуску моделювання.

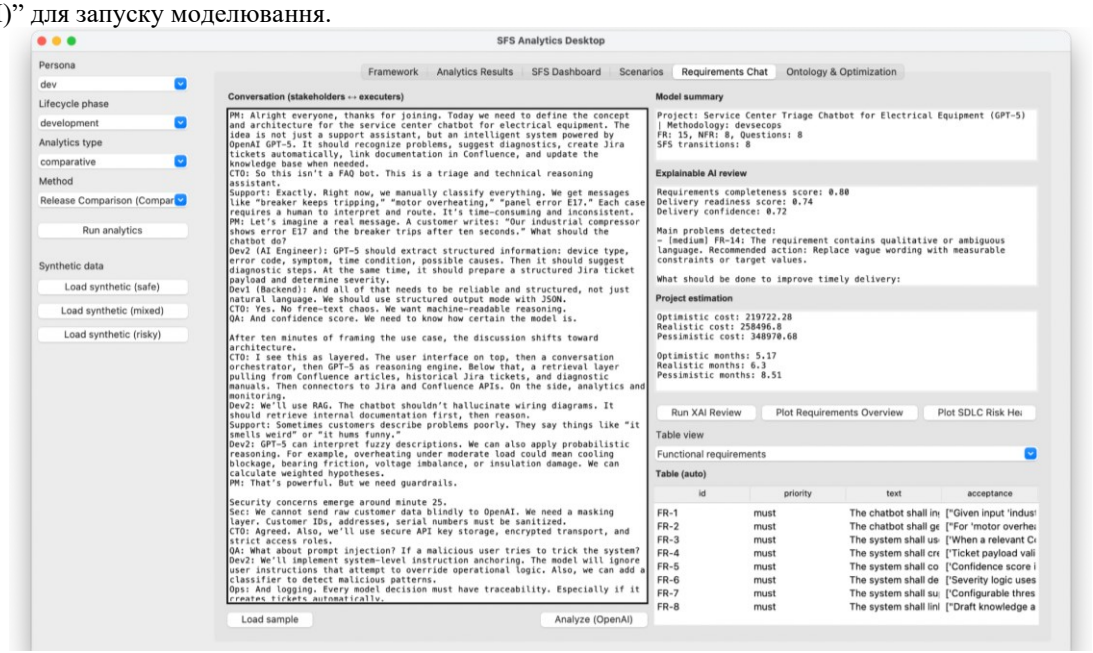


Рис. 8. Вкладка “Requirements Chat”

Права частина відображає результати. Блок “Model summary” показує узагальнені характеристики проєкту (методологія, кількість FR/NFR, SFS transitions). XAI review містить оцінки повноти вимог, готовності

до доставки та виявлені проблеми. Блок “Project estimation” подає “optimistic/realistic/pessimistic” оцінки вартості й тривалості. Внизу, доступні кнопки для XAI-аналізу та побудови графіків. Секція “Table view” дозволяє перемикати представлення (FR, NFR тощо), а таблиця відображає структуровані вимоги з атрибутами. Нижче доступні кнопки для XAI-аналізу та побудови графіків: “Plot Requirements Overview” (рис. 9) та “Plot SDLC Risk Heatmap” (рис. 10).

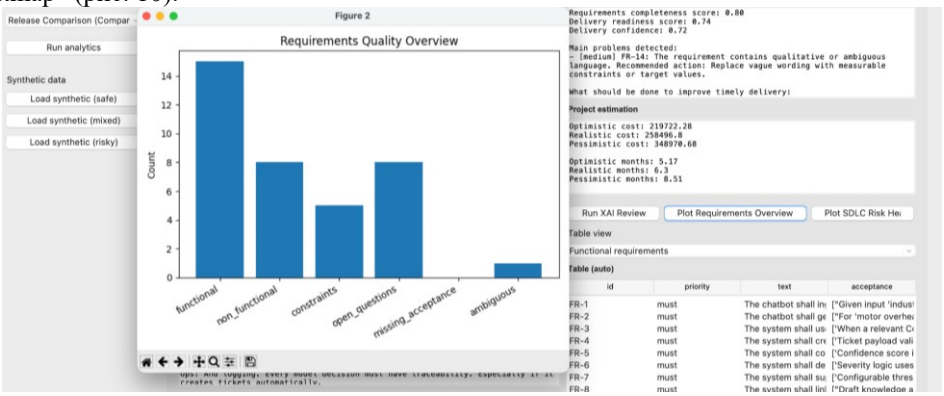


Рис. 9. Результат виконання кнопки “Plot Requirements Overview”

Кнопка “Plot Requirements Overview” будує узагальнену діаграму якості вимог на основі змодельованого JSON-представлення. На графіку відображено кількісний розподіл різних категорій: functional, non-functional, constraints, open_questions, а також індикатори проблем — missing_acceptance та ambiguous. Це дозволяє швидко оцінити структуру вимог і потенційні ризики. Наприклад, велика кількість функціональних вимог при наявності “missing_acceptance” сигналізує про неповноту критеріїв приймання. Категорія ambiguous вказує на розмиті формулювання, що можуть вплинути на строки delivery. Таким чином, діаграма виконує роль візуального XAI-інструменту: він не лише показує обсяг вимог, а й підсвічує проблемні зони, які потребують уточнення перед переходом до архітектури або оцінки бюджету.

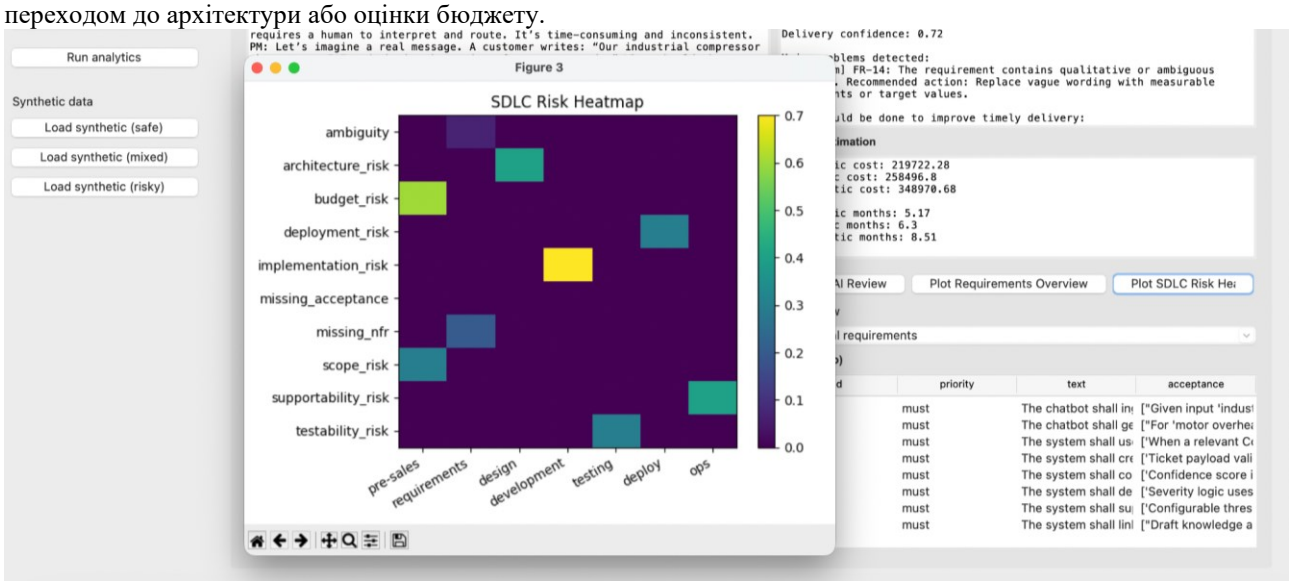


Рис. 10. Результат виконання кнопки “Plot SDLC Risk Heatmap”

Кнопка “Plot SDLC Risk Heatmap” будує теплову карту ризиків по фазах SDLC на основі змодельованих вимог і XAI-аналізу. По осі X відображено фази ЖЦ (pre-sales, requirements, design, development, testing, deploy, ops), а по осі Y — типи ризиків (ambiguity, budget_risk, architecture_risk, implementation_risk, missing_acceptance, missing_nfr, scope_risk, supportability_risk, testability_risk). Інтенсивність кольору показує рівень ризику (від 0 до ~0.7): темні зони — низький ризик, яскраві — підвищений. Наприклад, високий “implementation_risk” у фазі development сигналізує про технічну складність або неповні вимоги. “Budget_risk” у “pre-sales” вказує на ризик недооцінки вартості. “Missing_acceptance” або “missing_nfr” серед вимог означає загрозу затримок через неформалізовані критерії або нефункціональні вимоги. Таким чином, heatmap забезпечує візуальний огляд того, де саме у ЖЦПЗ концентруються ризики, і допомагає менеджеру або архітектору прийняти превентивні рішення (уточнення вимог, перегляд бюджету, додаткове тестування тощо).

Вкладка “Ontology & Optimization” (рис. 11) демонструє інтеграцію семантики, нечіткої логіки та оптимізації. Ліва частина містить шлях до OWL-онтології та текстові вимоги у спрощеному форматі. Після запуску формується запит на основі онтології (промпт, праворуч угорі), де показано, як вимоги інтерпретуються з урахуванням класів онтології (наприклад, “AcceptanceCriterion”, “Constraint”, “CostEstimate”). Блок “Feedback notes” відображає результати симуляції Монте Карло (середня вартість, P80, середня тривалість), оцінку ризику та підсумок генетичної оптимізації (кількість вибраних вимог, оптимізована цінність).

У таблиці результатів (Results table) показано нечітку квантифікацію якісних термінів (“very fast”, “secure”, “reliable”): кожен термін зіставляється з метрикою (performance, security, reliability), отримує числовий score (0–1) та пояснення. Таким чином, таблиця демонструє, як якісні формулювання трансформуються у вимірювані параметри для подальшої оцінки ризику й оптимізації.

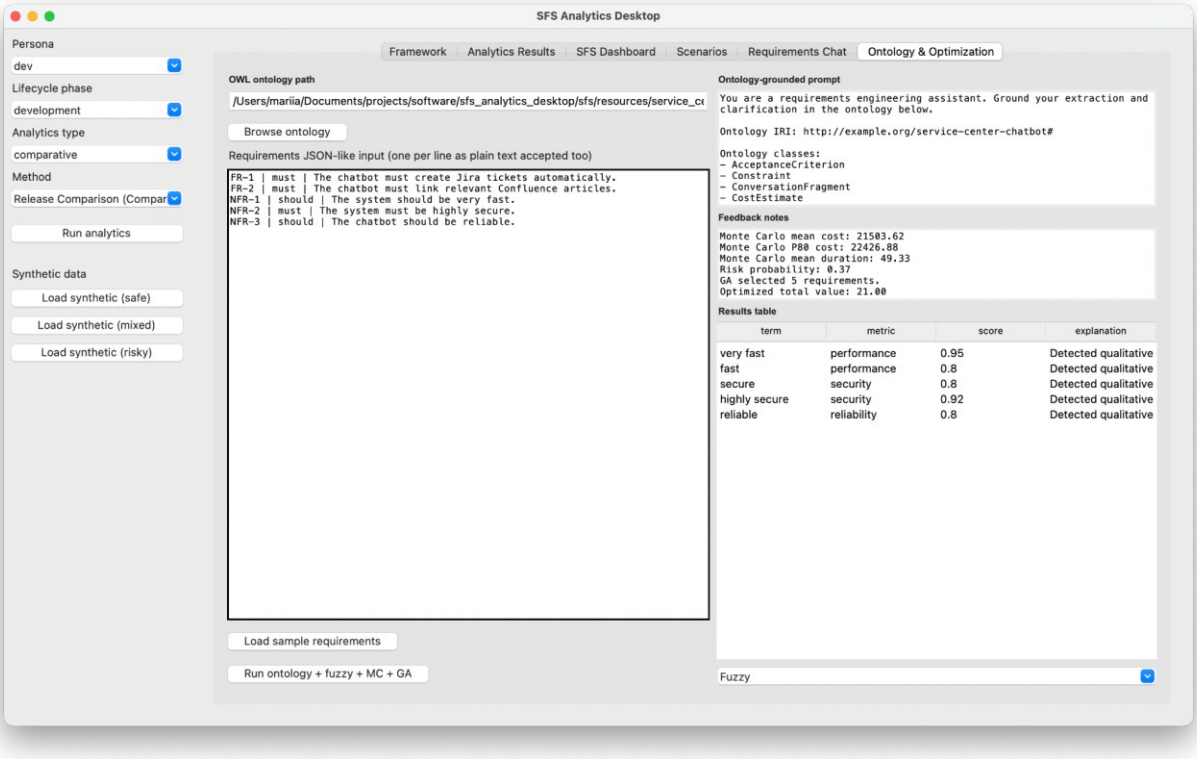


Рис. 11. Вкладка для роботи з онтологіями та оптимізацією вимог

Вкладка “Scenarios” (рис. 12) реалізує сценарій-орієнтовану аналітику для прийняття рішень у межах ЖЦПЗ. У верхній частині вибирається сценарій, наприклад, “architecture_selection”.

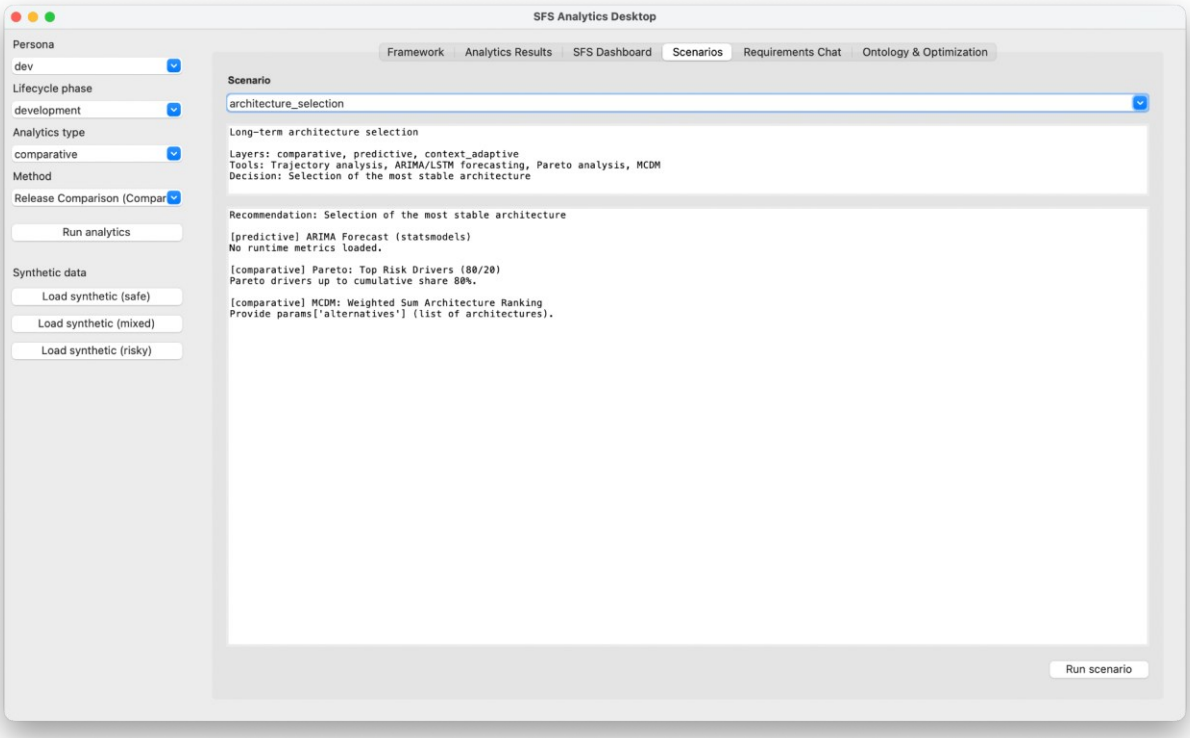


Рис. 12. Вкладка для роботи зі сценаріями

Формалізований опис архітектури включає:

- “layers” – аналітичні шари (comparative, predictive, context adaptive);

- “tools” – застосовані методи (ARIMA/LSTM, Pareto, MCDM тощо);
- “decision” – ціль управлінського рішення (вибір найстабільнішої архітектури).

Основне поле (рис. 12) показує результати виконання:

- прогнозна аналітика (ARIMA forecast);
- порівняльна (Pareto 80/20);
- багатокритеріальна оптимізація (MCDM).

Якщо дані або параметри (наприклад, перелік архітектур) не передані, ПЗ явно повідомляє про це. Кнопка “Run scenario” запускає комплексний сценарій, який інтегрує кілька методів та формує рекомендацію. У даному випадку щодо вибору архітектури з найменшим ризиком і найбільш стабільною траєкторією розвитку.

Вкладка “SFS Dashboard” (рис. 13) призначена для інтегрованого моніторингу ФСПЗ у динаміці. У верхній частині розміщені керуючі елементи:

- load CSV та Generate Synthetic — завантаження реальних або синтетичних даних;
- context weight boost — регулювання ваги контекстних сигналів у розрахунку SFS;
- відображення поточного State (R) та SFS index (0.632).

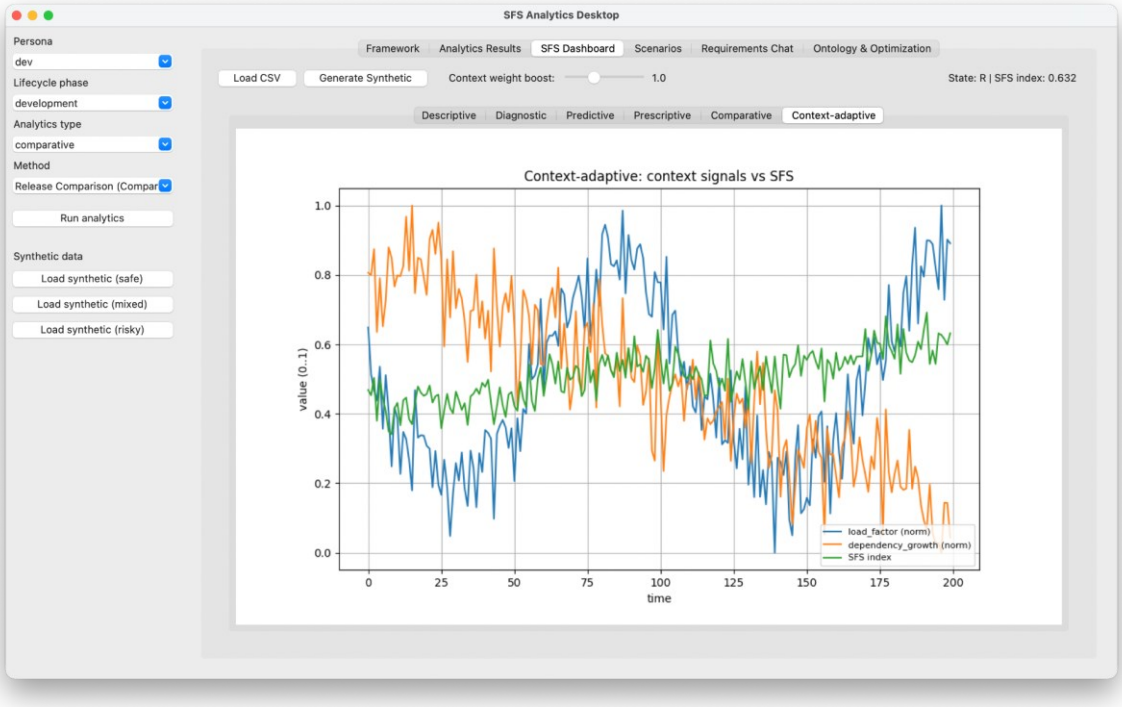


Рис. 13. Приладова панель ФСПЗ

Графік демонструє режим Context-adaptive. На ньому показано:

- синя крива — нормалізований фактор навантаження (load_factor);
- помаранчева — зростання залежностей (dependency_growth);
- зелена — інтегральний SFS index.

ФСПЗ формується як агрегована функція контекстних сигналів. Видно, що при зростанні “dependency_growth” і нестабільності “load_factor” “індекс SFS” змінюється відповідно, відображаючи адаптивну реакцію системи. Додаткові вкладки (Descriptive, Diagnostic, Predictive, Prescriptive, Comparative, Context-adaptive) дозволяють перемикатися між типами аналітики. Таким чином, приладова панель забезпечує стано-орієнтований контроль, аналіз трендів, виявлення деградацій та підтримку управлінських рішень у реальному часі.

Вкладка “Analytics Results” (рис. 14) призначена для аналізу завантажених датасетів та результатів аналітичних методів. У блоці “Summary” відображається метаінформація про датасет: ім’я файлу, шлях, кількість рядків (180), кількість колонок (11), числові та текстові поля. Це дозволяє швидко оцінити структуру даних перед аналізом. На формі доступні кнопки:

- “Load dataset” для завантаження CSV;
- “Use in bundle” для інтеграції датасету в ФСПЗ-модель;
- “Plot dataset” для побудова графіка;
- “Show stats” для описової статистики.

Таблиця відображає числові колонки (наприклад, vuln_count, cvss_agg, latency, throughput) для швидкого перегляду значень.

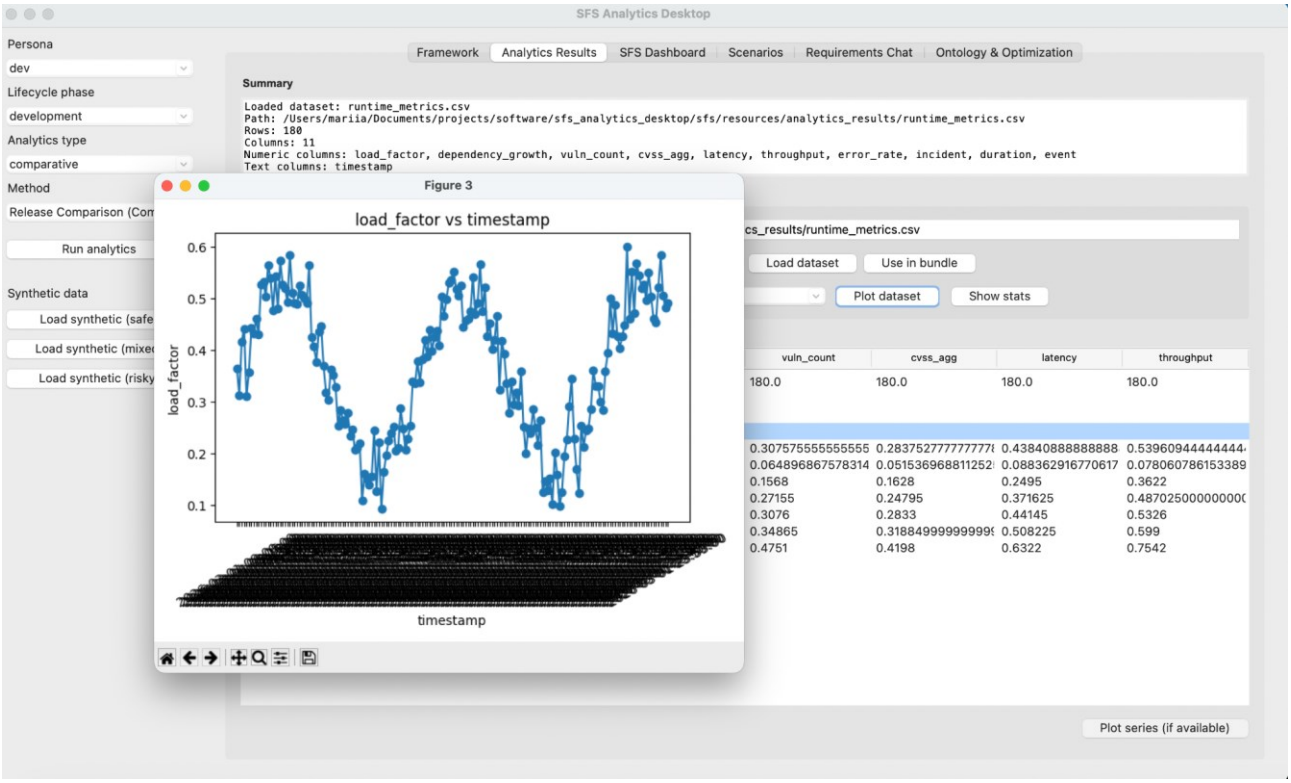


Рис. 14. Форма для детальнішого аналізу результатів спостережень

На прикладі графіка “load_factor vs timestamp” показано часову динаміку навантаження системи. Видно періодичні хвилеподібні коливання між ~0.1 і ~0.6, що може відповідати циклам використання або зміні трафіку. Такі тренди важливі для прогнозування деградації, виявлення піків навантаження та оцінки впливу на ФСПЗ-індекс.

Висновки з даного дослідження і перспективи подальших розвідок у даному напрямі

Розроблене ПЗ являє собою комплексну багаторівневу систему аналітики ФСПЗ, яка інтегрує описову, діагностичну, прогнозну, прескриптивну, порівняльну та контекстно-адаптивну аналітику в єдиному середовищі. Ключовою особливістю є перехід від аналізу ізольованих метрик до моделювання ПЗ як динамічної системи станів із переходами, що пов’язані з рішеннями на різних етапах ЖЦПЗ.

Розроблені засоби аналітики забезпечують:

- інтеграцію аналітики вимог (LLM + XAI + оцінка ризиків);
- формалізацію ЖЦПЗ через моделі ФСПЗ;
- онтологічне узгодження знань (OWL → prompt grounding);
- стохастичне моделювання (Монте Карло, Моделі Маркова);
- оптимізацію рішень (ГА);
- багатокритеріальний вибір архітектурних альтернатив;
- візуалізацію ризиків та контекстних сигналів.

Таким чином, створено керований аналітичний контур підтримки рішень для DevSecOps- та проєктів на основі ШІ. Функціональність системи тестувалася на:

- підготовлених структурованих датасетах (runtime-метрики, ЖЦПЗ-ризики, вимоги);
- синтетичних наборах даних із керованими режимами ризику (safe / mixed / risky);
- змодельованих діалогах для перевірки модуля “Requirements Intelligence”;
- онтологіях різної складності (smoke-test та розширена модель).

Такий підхід дозволив:

- перевірити стабільність алгоритмів;
- протестувати крайні сценарії;
- відтворити контрольовані ризикові ситуації;
- провести валідацію візуалізацій та пояснювальних механізмів.

Використання синтетичних даних є обґрунтованим на етапі прототипування, оскільки забезпечує контроль над розподілами, шумом, переходами станів і дозволяє тестувати алгоритми без обмежень конфіденційності.

До наукової новизни роботи можна віднести:

- Формалізацію ФСПЗ як інтеграційної аналітичної сутності, що об’єднує різнорідні дані з усіх фаз ЖЦ в єдину стано-орієнтовану модель.
- Запропоновано багаторівневу структуру аналітики, узгоджену з моделлю ФСПЗ та переходів між ними, на відміну від існуючих підходів, які аналізують окремі метрики або ізольовані напрями (MSR, defect

prediction, runtime-моніторинг, технічний борг).

- Удосконалено класифікацію типів аналітики ПЗ шляхом їх систематизації відповідно до ролі у формуванні, інтерпретації та прогнозуванні змін ФСПЗ.
- Набуло подальшого розвитку підхід щодо переходу від метрико-орієнтованого до стано-орієнтованого підходу, що забезпечує цілісну оцінку еволюції ПЗ.
- Запропонований підхід створює теоретичну основу для інтеграції аналітики вимог, архітектури, реалізації та експлуатації у єдину систему підтримки прийняття рішень. Це забезпечує можливість переходу від фрагментованого моніторингу до інтелектуального управління ЖЦПЗ.
- Серед напрямків майбутніх досліджень слід вважати:
- Валідацію на реальних індустріальних даних, на основі інтеграції з реальними системами для емпіричної перевірки ФСПЗ-моделі.
- Розробку уніфікованої метамоделі, що усуне розриви між вимогами, кодом, тестами та runtime-даними — єдиної моделі знань про програмний продукт.
- Калібрування ФСПЗ-індексу на основі порівняння з фактичними інцидентами та релізними провалами.
- Розширення XAI-механізмів для пояснення прогнозних та оптимізаційних рішень.
- Федеративну аналітику та MLOps.

Запропонований підхід демонструє можливість переходу від фрагментарного аналізу метрик до інтегрованої, стано-орієнтованої, пояснюваної та оптимізаційної аналітики ЖЦПЗ. Отримані результати підтверджують технічну реалізованість концепції, а подальша перевірка на реальних даних дозволить оцінити її практичну ефективність та індустріальну застосовність.

Література

1. Caldeira, J., Brito e Abreu, F., Cardoso, J. et al. Software Development Analytics in Practice: A Systematic Literature Review. Arch Computat Methods Eng 30, 2041–2080 (2023). https://doi.org/10.1007/s11831-022-09864-y.
2. Laiq, M., Ali, N. B., Börstler, J., & Engström, E. (2024). Software analytics for software engineering: A tertiary review. arXiv preprint arXiv:2410.05796. https://doi.org/10.48550/arXiv.2410.05796.
3. D. Zhang, S. Han, Y. Dang, J.G. Lou, H. Zhang and T. Xie, "Software Analytics in Practice," in IEEE Software, vol. 30, no. 5, pp. 30-37, Sept.-Oct. 2013, https://doi.org/10.1109/MS.2013.94.
4. A. E. Hassan, "Predicting faults using the complexity of code changes," 2009 IEEE 31st International Conference on Software Engineering, Vancouver, BC, Canada, 2009, pp. 78-88, doi: 10.1109/ICSE.2009.5070510.
5. Vidoni, M. (2022). A systematic process for mining software repositories: Results from a systematic literature review. Information and Software Technology, 144, 106791. https://doi.org/10.1016/j.infsof.2021.106791.
6. Kagdi, H., Collard, M. L., & Maletic, J. I. (2007). A survey and taxonomy of approaches for mining software repositories in the context of software evolution. *Journal of Software Maintenance and Evolution: Research and Practice, 19*(2), 77–131. https://doi.org/10.1002/smr.344.
7. Soliman, M., Albonico, M., Malavolta, I., & Wortmann, A. (2025). Mining software repositories for software architecture — A systematic mapping study. Information and Software Technology, 181, 107677. https://doi.org/10.1016/j.infsof.2025.107677.
8. Kadebu, N., Sabri, S., & Idris, N. (2022). Requirements classification considering maintainability as a security requirement. SoftwareX, 20, 101268. https://doi.org/10.1016/j.softx.2022.101268.
9. Landauer, M., Onder, S., Skopik, F., & Wurzenberger, M. (2023). Deep learning for anomaly detection in log data: A survey. Machine Learning with Applications, 12, 100470. https://doi.org/10.48550/arXiv.2207.03820.
10. Khan, Z.A., Shin, D., Bianculli, D. et al. Impact of log parsing on deep learning-based anomaly detection. Empir Software Eng 29, 139 (2024). https://doi.org/10.1007/s10664-024-10533-w.
11. Sánchez, C., Schneider, G., Ahrendt, W., Bartocci, E., Bianculli, D., Colombo, C., ... & Weiss, A. (2019). A survey of challenges for runtime verification from advanced application domains (beyond software). Formal Methods in System Design, 54(3), 279-335. https://doi.org/10.48550/arXiv.1811.06740.
12. Taleb, R., Hallé, S., & Khoury, R. (2023). Uncertainty in runtime verification: A survey. Computer Science Review, 50, 100594. https://doi.org/10.1016/j.cosrev.2023.100594.
13. Lyashkevych, M.Y., Lyashkevych, V.Y., & Shuvar, R.Y. (2025). Definition and Formalization of the Software Functional State Concept Throughout the Development Life Cycle. Electronics and Information Technologies, 32, 151–170. https://doi.org/10.30970/eli.32.11.
14. S. Hosseini, B. Turhan and D. Gunarathna, "A Systematic Literature Review and Meta-Analysis on Cross Project Defect Prediction," in IEEE Transactions on Software Engineering, vol. 45, no. 2, pp. 111-147, 1 Feb. 2019. https://doi.org/10.1109/TSE.2017.2770124.
15. Bhutamapuram, U. S., Chonari, F., Anilkumar, G. K., & Konchada, S. K. (2025). LLMs for defect prediction in evolving datasets: Emerging results and future directions. In Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering (FSE Companion '25) (pp. 520–524). Association for Computing Machinery. https://doi.org/10.1145/3696630.3728491.
16. G. Kånåhols, S. Hasan and P. Erik Strandberg, "Integrating Time Series Anomaly Detection Into DevOps Workflows," in IEEE Access, vol. 13, pp. 46459-46477, 2025. https://doi.org/10.1109/ACCESS.2025.3550665.
17. Geçer, B. G., & Tarhan, A. K. (2025). Explainable AI Framework for Software Defect Prediction. Journal of

Software: Evolution and Process, 37(4), e70018. https://doi.org/10.1002/smr.70018.

18. Lenarduzzi, V., Besker, T., Taibi, D., Martini, A., & Fontana, F. A. (2019). Technical debt prioritization: State of the art. A systematic literature review. arXiv preprint arXiv:1904.12538. https://doi.org/10.48550/arXiv.1904.12538.

19. Li, Z., Avgeriou, P., & Liang, P. (2015). A systematic mapping study on technical debt and its management. Journal of Systems and Software, 101, 193–220. https://doi.org/10.1016/j.jss.2014.12.027.

20. Moulla, M., Lenarduzzi, V., Taibi, D., & Palomba, F. (2023). Technical debt measurement: An exploratory literature review. In Proceedings of the International Workshop on Technical Debt (pp. 1–10). CEUR-WS. https://ceur-ws.org/Vol-3852/paper3.pdf.

21. Liao, L., Li, H., Shang, W., Sporea, C., Toma, A., & Sajedi, S. (2023). Adapting performance analytic techniques in a real-world database-centric system: An industrial experience report. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2023)* (pp. 1855–1866). Association for Computing Machinery. https://doi.org/10.1145/3611643.3613893.

22. М. Ляшкевич (2025). гібридний інтелектуальний фреймворк для покращення точності збору та повноти формалізації вимог до програмного забезпечення. Матеріали 2-ї Міжнародної науково-практичної інтернет-конференції “Подолання кордонів: розкриття динаміки передових досліджень та їх трансформаційного впливу на глобальну сферу”, Дніпро, Україна, Липень 11-12, 2025 - Дніпро, 85-87. Url: http://www.wayscience.com/wp-content/uploads/2025/07/Conference-Proceedings-July-10-11-2025-1.pdf.

References

1. Caldeira, J., Brito e Abreu, F., Cardoso, J. et al. Software Development Analytics in Practice: A Systematic Literature Review. Arch Computat Methods Eng 30, 2041–2080 (2023). https://doi.org/10.1007/s11831-022-09864-y.

2. Laiq, M., Ali, N. B., Börstler, J., & Engström, E. (2024). Software analytics for software engineering: A tertiary review. arXiv preprint arXiv:2410.05796. https://doi.org/10.48550/arXiv.2410.05796.

3. D. Zhang, S. Han, Y. Dang, J.G. Lou, H. Zhang and T. Xie, "Software Analytics in Practice," in IEEE Software, vol. 30, no. 5, pp. 30-37, Sept.-Oct. 2013, https://doi.org/10.1109/MS.2013.94.

4. A. E. Hassan, "Predicting faults using the complexity of code changes," 2009 IEEE 31st International Conference on Software Engineering, Vancouver, BC, Canada, 2009, pp. 78-88, doi: 10.1109/ICSE.2009.5070510.

5. Vidoni, M. (2022). A systematic process for mining software repositories: Results from a systematic literature review. Information and Software Technology, 144, 106791. https://doi.org/10.1016/j.infsof.2021.106791.

6. Kagdi, H., Collard, M. L., & Maletic, J. I. (2007). A survey and taxonomy of approaches for mining software repositories in the context of software evolution. *Journal of Software Maintenance and Evolution: Research and Practice, 19*(2), 77–131. https://doi.org/10.1002/smr.344.

7. Soliman, M., Albonico, M., Malavolta, I., & Wortmann, A. (2025). Mining software repositories for software architecture — A systematic mapping study. Information and Software Technology, 181, 107677. https://doi.org/10.1016/j.infsof.2025.107677.

8. Kadebu, N., Sabri, S., & Idris, N. (2022). Requirements classification considering maintainability as a security requirement. SoftwareX, 20, 101268. https://doi.org/10.1016/j.softx.2022.101268.

9. Landauer, M., Onder, S., Skopik, F., & Wurzenberger, M. (2023). Deep learning for anomaly detection in log data: A survey. Machine Learning with Applications, 12, 100470. https://doi.org/10.48550/arXiv.2207.03820.

10. Khan, Z.A., Shin, D., Bianculli, D. et al. Impact of log parsing on deep learning-based anomaly detection. Empir Software Eng 29, 139 (2024). https://doi.org/10.1007/s10664-024-10533-w.

11. Sánchez, C., Schneider, G., Ahrendt, W., Bartocci, E., Bianculli, D., Colombo, C., ... & Weiss, A. (2019). A survey of challenges for runtime verification from advanced application domains (beyond software). Formal Methods in System Design, 54(3), 279-335. https://doi.org/10.48550/arXiv.1811.06740.

12. Taleb, R., Hallé, S., & Khoury, R. (2023). Uncertainty in runtime verification: A survey. Computer Science Review, 50, 100594. https://doi.org/10.1016/j.cosrev.2023.100594.

13. Lyashkevych, M.Y., Lyashkevych, V.Y., & Shuvar, R.Y. (2025). Definition and Formalization of the Software Functional State Concept Throughout the Development Life Cycle. Electronics and Information Technologies, 32, 151–170. https://doi.org/10.30970/eli.32.11.

14. S. Hosseini, B. Turhan and D. Gunarathna, "A Systematic Literature Review and Meta-Analysis on Cross Project Defect Prediction," in IEEE Transactions on Software Engineering, vol. 45, no. 2, pp. 111-147, 1 Feb. 2019. https://doi.org/10.1109/TSE.2017.2770124.

15. Bhutamapuram, U. S., Chonari, F., Anilkumar, G. K., & Konchada, S. K. (2025). LLMs for defect prediction in evolving datasets: Emerging results and future directions. In Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering (FSE Companion ’25) (pp. 520–524). Association for Computing Machinery. https://doi.org/10.1145/3696630.3728491.

16. G. Kånåhols, S. Hasan and P. Erik Strandberg, "Integrating Time Series Anomaly Detection Into DevOps Workflows," in IEEE Access, vol. 13, pp. 46459-46477, 2025. https://doi.org/10.1109/ACCESS.2025.3550665.

17. Geçer, B. G., & Tarhan, A. K. (2025). Explainable AI Framework for Software Defect Prediction. Journal of Software: Evolution and Process, 37(4), e70018. https://doi.org/10.1002/smr.70018.

18. Lenarduzzi, V., Besker, T., Taibi, D., Martini, A., & Fontana, F. A. (2019). Technical debt prioritization: State of the art. A systematic literature review. arXiv preprint arXiv:1904.12538. https://doi.org/10.48550/arXiv.1904.12538.

19. Li, Z., Avgeriou, P., & Liang, P. (2015). A systematic mapping study on technical debt and its management. Journal of Systems and Software, 101, 193–220. https://doi.org/10.1016/j.jss.2014.12.027.

20. Moulla, M., Lenarduzzi, V., Taibi, D., & Palomba, F. (2023). Technical debt measurement: An exploratory literature review. In Proceedings of the International Workshop on Technical Debt (pp. 1–10). CEUR-WS. https://ceur-ws.org/Vol-3852/paper3.pdf.

21. Liao, L., Li, H., Shang, W., Sporea, C., Toma, A., & Sajedi, S. (2023). Adapting performance analytic techniques in a real-world database-centric system: An industrial experience report. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2023)* (pp. 1855–1866). Association for Computing Machinery. https://doi.org/10.1145/3611643.3613893.

22. M. Lyashkevycha (2025). hibrydnyy intelektual′nyy freymvork dlya pokrashchennya tochnosti zboru ta povnoty formalizatsiyi vymoh do prohramnoho zabezpechennya. Materialy 2-yi Mizhnarodnoyi naukovo-praktychnoyi internet-konferentsiyi “Podolannya kordoniv: rozkryttya dynamiky peredovykh doslidzhen′ ta yikh transformatsiynoho vplyvu na hlobal′nu sferu”, Dnipro, Ukrayina, 11-12 lypnya 2025 r. - Dnipro, 85-87. URL-adresa: http://www.wayscience.com/wp-content/uploads/2025/07/Conference-Proceedings-July-10-11-2025-1.pdf.