Issue 18, part 1, 2025

https://doi.org/10.23939/sisn2025.18.260

УДК 004.45:004.42

ФОРМУВАННЯ ТЕХНІЧНОЇ ДОКУМЕНТАЦІЇ ІТ ПРОЄКТІВ В КОНТЕКСТІ РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Олег Захарія, Наталія Кунанець, Юрій Жовнір

Національний університет «Львівська політехніка»,
кафедра інформаційних систем та мереж, Львів, Україна
E-mail: ozakhar@gmail.com, ORCID: 0009-0008-6979-129X
E-mail: nek.lviv@gmail.com, ORCID: 0000-0003-3007-2462
E-mail: zhovnir@astra.in.ua, ORCID: 0009-0006-6186-2861

Анотація. У статті досліджено процеси формування технічної документації ІТ проєктів з розроблення програмного забезпечення із урахуванням особливостей методології Agile та DevOps. Запропоновано формалізми побудови системи документування як інтегрованої складової життєвого циклу ІТ-проєкту. Розглянуто етапи трансформації користувацької історії у структуровані документи, роль безпекових шаблонів, прототипування та програмного коду як інформаційних джерел для генерації специфікацій. Акцент зроблено на автоматизованому документуванні, включно з автогенерацією API-документації, тестових звітів та використанням матриць трасування вимог. Проаналізовано взаємозв'язки документів та етапів проєкту (епіки, коміти, тести) у контексті CI/CD-конвеєра. Подано формалізми та сформовано на їх основі модель, діаграми та підходи, які використані при формуванні відповідної архітектури, що в системному поєднанні забезпечують повноту, актуальність і структурованість технічної документації на програмне забезпечення.

Ключові слова: технічна документація, програмне забезпечення, користувацька історія, епіки, безпекові шаблони, прототипування, формалізація, CI/CD, Agile, DevOps, API-документація, трасування вимог, автогенерація документації, документація, що сформована на основі коду

Постановка проблеми

В умовах стрімкого розвитку технологій та розроблення програмного забезпечення, дедалі більшого поширення методологій управління проєктами Agile та DevOps, питання створення технічної документації на ІТ проєкти в контексті щораз ширшої автоматизації цих процесів набуває особливої актуальності. Технічна документація, що охоплює специфікації вимог, архітектурні рішення, особливості API-інтерфейсів, інструкції з розгортання, тестові сценарії та експлуатаційні настанови, відіграє критичну роль у забезпеченні якості, узгодженості та масштабованості програмного продукту на всіх етапах його життєвого циклу. Незважаючи на розвиток інструментів безперервної інтеграції та доставки, проблема формалізації й інтеграції станів документування програмного забезпечення у ці процеси залишається відкритою, особливої ваги це набуває для невеликих команд, а також для процесів створення безпеково чутливих інформаційних систем.

Відсутність актуальної, структурованої та трасованої документації ускладнює реалізацію складних функціональних вимог, уповільнює адаптацію нових учасників команди, підвищує ризики розробки, а також знижує ефективність тестування та підтримки. Водночас розвиток інтелектуальних помічників,

розширення можливостей репозиторіїв коду та впровадження підходу «документація як код» створюють передумови для повної або часткової автоматизації процесів документування розроблення та супроводу програмних продуктів. Актуальність дослідження обумовлюється необхідністю інтегрування технічної документації в процеси розроблення, зокрема автоматизувати її створення, забезпечити відповідність міжнародним стандартам (IEEE 830, ISO/IEC 26514 та ін.), гарантувати її актуальність і трасованість у межах CI/CD-конвеєра. Особливо важливою є формалізація процесів формування документації на основі історій користувачів, епіків, безпекових шаблонів та прототипів, що дозволяє систематизувати знання про продукт, зробити їх доступними, перевіреними та відтворюваними.

Аналіз останніх досліджень та публікацій

У сучасній практиці створення технічної документації IT проєкту із створення прогамного забезпечення розглядається не лише як його супровідна частина, а як інтегрований компонент життєвого циклу ПЗ, що забезпечує структурованість, трасованість та довготривалу підтримку. A. Aimar (2001) відзначає, що документація необхідна на всіх етапах розроблення програмного продукту: від етапу визначення вимог користувача, через проєктування та реалізацію, до доставки і супроводу продукту. З цієї причини документація повинна бути належним чином спроєктована, створена, опублікована та підтримувана. На думку Balasubramaniam et al (2010) у командах, які працюють за Agile-методологією, технічна документація створюється поступово, змінюючи свою форму у процесі ітерацій, за принципом "just-enough documentation", що дозволяє зберегти належний баланс між гнучкістю та структурою. Такий підхід дозволяє уникнути надмірної забюрократизованості, водночас забезпечуючи необхідний рівень фіксації вимог, архітектури та проєктних рішень. Behutiye et al (2022) вважають, що аджайл-розробка програмного забезпечення (Agile Software Development, далі ASD) сприяє мінімізації обсягів супроводжуючої документації та часто надає перевагу функціональним вимогам над вимогами щодо якості (Quality Requirements, далі QRs). Такий підхід до мінімізації обсягів проєктної документації може бути корисним в контексті скорочення часу виходу продукту на ринок. На процеси документування суттєво впливають часові обмеження, рівень обізнаності щодо QRs та комунікаційні проблеми. Відсутність або застарілість документації на вимоги до рівня якості може призвести до технічного відставання та цілісного спільного розуміння цих вимог. Запропонована модель базується на емпіричних знаннях щодо практик документування QRs в ASD. Synko A., & Peleshchyshyn, A. (2020) вважають, що у зв'язку зі стрімким розвитком різнопланових проєктів із створення програмних продуктів, застосунків та інформаційних систем, виникає потреба у підготовці значних обсягів технічної документації, в якій подається опис створеного програмного продукту з різних точок зору. Дослідниками подано класифікацію типів документації, зокрема, для документування вимог, що визначають очікування до програмного забезпечення. У свою чергу, дослідження Kolomiiets, I. (2024) демонструє, що документація повинна бути інтегрована у DevOps-конвеєр, підтримуючи автоматичне оновлення, перевірку та версійність на кожному етапі CI/CD, оскільки технічна документація в методології DevOps - це не просто ведення записів; це основа успішних практик, що реалізуються за цією методологією. Вона допомагає роботі DevOps команд, підвищує комфортність процесів співпраці розробників та забезпечує якісне управління повним життєвим циклом розробки. Від внутрішньої документації до посібників з API, документація консолідує колективні знання проєктної команди, роблячи складні програмні системи зрозумілими та комфортно керованими. На думку дослідників, створення якісної документації на програмне забезпечення відіграє вирішальну роль у методології DevOps з кількох причин. Якісна та повна документація спрощує процеси підтримки узгодженості протягом усього життєвого циклу проекту, від розробки до розгортання і супроводу. Buchgeher et al. (2018) вважають, що проєкти, в яких документація є фрагментарною або застарілою, потребують значно більше часу для підтримки та модифікацій. Без належного документованого опису до 30–50 % зусиль проєктної команди витрачається на з'ясування логіки функціонування компонентів. У статті представлено один із підходів до авто-

матичного створення технічної документації. Brockmann, R. J. (1990) розроблено підходи до формування чіткої та точної документації. Low et al. (1995) вважають, що підготовка користувацької документації» допомагає читачеві отримати знання про процеси її написання, від початкового формування до етапу редагування. Rai, A., Kumar, A., & Singh, P. (2022) відзначають, що документування коду вважається однією з найкращих практик у процесах розробки програмного забезпечення, яка вимагає докладання значних зусиль розробниками. У поданому дослідниками огляді проаналізовані сучасні практики документування коду. Ries, R. (1990) виділяє положення «Стандарту документації користувача програмного забезпечення» IEEE Std 1063-1990, розробленого робочим комітетом Інституту інженерів з електротехніки та електроніки (IEEE), та можливості його використання як інструменту для розроблення структури документації користувача програмного забезпечення. Blome M. (2025) наголошує, що вимоги до технічної документації зростають, контент слід створювати швидше, послідовно підтримувати та доставляти користувачам в різних форматах. Таким чином, автоматизація процесів створення технічної документації знижує ризики використання застарілої інформації, мінімізує ручну працю, підвищує якість проєктів та забезпечує відповідність актуальним вимогам, що висуваються до програмного продукту.

Формулювання цілі статті

Метою статті є дослідити концептуальні засади та практичні аспекти створення технічної документації в рамках повного життєвого циклу розробки сучасного програмного забезпечення, запропонувати моделі її побудови, розглянути можливості автоматизації її формування та супроводу, а також оцінити потенціал впровадження конвеєрів CI/CD для системного забезпечення актуальності, повноти й структурованості технічних артефактів.

Виклад основного матеріалу

Технічна документація на ІТ проєкт − це консолідована система документів, яка супроводжує повний життєвий цикл програмного забезпечення та ІТ проєкту в цілому, містить детальну інформацію про процеси розроблення, впровадження, експлуатації, тестування, модернізації та супроводу. Вона відіграє ключову роль у забезпеченні прозорості та узгодженості розробницьких дій, сприяє ефективній взаємодії між членами команди, зменшує ризики виникнення помилок і спрощує процеси подальшого обслуговування та масштабування системи. До складу технічної документації, як правило, входять функціональні та нефункціональні вимоги до інформаційної системи, описи архітектури, специфікації інтерфейсів, інструкції з розгортання та конфігурації, API-документація, тест-кейси та звіти з тестування, інструкції з використання, опис системної інфраструктури, а також звіти про виявлені дефекти та заходи з усунення вразливостей. Позначимо D_{tech} − технічна документація ІТ проєкту; $d_i \in D_{tech}$ − окремий документ або артефакт; $t_i \in T$− етап життєвого циклу; $A(t_i) \subseteq D_{tech}$− множина артефактів, що створюються або оновлюються на етапі t_i. Подамо технічну документацію на програмний продукт як сукупність документів або артефактів:

$$D_{tech} = \{d_1, d_2, \dots d_n\} \quad (1)$$

Водночас подамо компонентний склад технічної документації на ІТ проєкт:

$$D_{\text{tech}} = D_{\text{req}} \cup D_{\text{arch}} \cup D_{\text{intf}} \cup D_{\text{deploy}} \cup D_{\text{api}} \cup D_{\text{test}} \cup D_{\text{user}} \cup D_{\text{infra}} \cup D_{\text{sec}}, \quad (2)$$

де D_{req} − вимоги до інформаційної системи $D_{\text{req}} = D_f \cup D_{nf}$; D_f − функціональні вимоги фіксують відомості про те, що саме повинна робити система; D_{nf} − нефункціональні вимоги − визначають як саме повинна працювати система (мова йде про продуктивність, безпеку, масштабованість тощо); D_{arch} − опис архітектури інформаційної системи; D_{intf} − специфікації інтерфейсів; D_{deploy} − інструкції з розгортання та конфігурування; D_{api} − API-документація; D_{test} − тестова документація $D_{\text{test}} = \{\text{TestCases}, \text{TestReports}\}$; D_{user} − інструкції з використання; D_{infra} − опис системної інфраструктури; D_{sec} − звіти про дефекти та заходи з безпеки $D_{\text{sec}} = \{\text{DefectReports}, \text{VulnerabilityFixes}\}$. При цьому D_{test} − підмножина технічної документації, яка охоплює всі артефакти, пов'язані з процесом тестування програмного забезпечення,

TestCases − тест-кейси є по суті структурованими сценаріями перевірки поведінки системи. Кожен тест-кейс описує вхідні дані, передумови, очікувані результати, фактичну поведінку, статус (пройдено/непройдено). TestReports − звіти про тестування є документами, що містять результати виконання тестів. Множина D_{test} включає тест-кейси та звіти про тестування. Тест-кейси описують що і як перевіряється, а звіти про тестування подають результати, отримані в процесі тестування. Тест-кейси та звіти про тестування є основою для контролю якості ПЗ, трасування вимог, аналізу дефектів і планування регресивного тестування. D_{sec} − множина безпекової документації, як частина технічної документації IT проєкту, що фокусується на виявленні, описі та усуненні вразливостей і дефектів безпеки в системі, DefectReports − звіти про дефекти це документи, що містять зафіксовані помилки або збої в системі. В контексті безпеки − це можуть бути некоректна автентифікація, відсутність перевірки прав доступу, XSS, SQL Injection тощо. Вони зазвичай включають ідентифікатор дефекту; опис проблеми; умови відтворення; рівень критичності; дату виявлення та ідентифікатор призначеної особи. VulnerabilityFixes − опис усунення вразливостей, тобто документи або записи, які містять спосіб усунення знайденої вразливості (наприклад, зміни в коді, патчі); технічне пояснення застосованого рішення; коміт до репозиторію або посилання на оновлення; тести, що підтверджують усунення; пов'язані вимоги або user stories, що були оновлені. Множина D_{sec} містить документацію, пов'язану з інцидентами безпеки, виявленими у програмному забезпеченні. Це є як звіти про виявлені проблеми, так і рішення або заходи, вжиті для їх усунення. Така документація є ключовою для забезпечення прозорості, аудиту, відповідності вимогам (наприклад, ISO/IEC 27001) та мінімізації ризиків повторного виникнення вразливостей. Функції належності документів до певної категорій можемо подати:

$$\forall d_i \in D_{\text{tech}}, \exists\, C_{\text{j}}: d_i \in C_{\text{j}},\ C_{\text{j}} \in \{D_{\text{req}}, D_{\text{arch}}, \ldots . D_{sec}\}\,, \tag{3}$$

де $\forall d_i \in D_{\text{tech}}$ для кожного елемента d_i, який належить до множини технічної документації D_{tech}, тобто для кожного документа в системі; $\exists\ C_{\text{j}}$ − існує така категорія документації C_{j}, що $d_i \in C_{\text{j}}$ − цей документ d_i належить до вказаної категорії C_{j}; $C_{\text{j}} \in \{D_{\text{req}}, D_{\text{arch}}, \ldots . D_{sec}\}$ − ця категорія належить до заздалегідь визначеного набору документації. Формальний запис стверджує, що кожен документ, який входить до технічної документації, має належати до однієї з формально визначених категорій. Особливо важливим є створення технічної документації від самого зародження проєкту, тобто від моменту формулювання ідеї. Саме на ранніх етапах − під час збору вимог, формулювання цілей, визначення ключових користувачів і сценаріїв використання − закладаються основи для побудови структурованої документації. Рання фіксація рішень і обґрунтувань дозволяє уникнути неузгодженостей на подальших етапах розроблення проєкту, забезпечує трасування змін і служить цінним джерелом знань для нових учасників команди або зовнішніх аудиторів. Наприклад, нехай d_i = "Тестовий звіт", тоді $C_j = D_{test}$ і $d_i = D_{\text{test}}$ і справджується:

$$\forall d_i = \text{Тестовий звіт} \in D_{tech}, \exists D_{test}: d_i \in D_{test} \in \{\ldots\} \tag{4}$$

Цей запис може бути корисним для реалізації етапу формального контролю повноти документації або автоматизованої класифікації документів у складних IT проєктах. В разі визначення кількісної оцінки припустимо, що $|\ D_{\text{tech}}|$=n − загальна кількість артефактів документації, тоді:

$$\text{n} = \sum_{j=1}^{k} |\ C_j\ |,$$

де C_j − категорія документації; k − кількість категорій, як описано вище.

Таке формалізоване подання дозволяє чітко розуміти процеси структурування технічної документації з метою подальшого автоматичного аналізу, генерації або перевірки на повноту. Таким чином, наявність якісної технічної документації є не лише абстрактною вимогою до проєктів з розроблення програмного забезпечення, а й важливим інструментом управління знаннями, підтримки якості продукту та забезпечення його довготривалої життєздатності. Підготовка та ведення технічної документації є одним з ключових компонентів реалізації проєкту, а процес її розроблення регламентується міжнародними стандартами, зокрема IEEE 830 (Software Requirements Specification), IEEE 1016 (Software Design Description), ISO/IEC 26514:2008 (Documentation for users of systems and software).

На рис. 1 зображено зв’язки між основними категоріями технічної документації (системна, користувацька, API, тестова) та фазами життєвого циклу розробки ІТ проєкту.

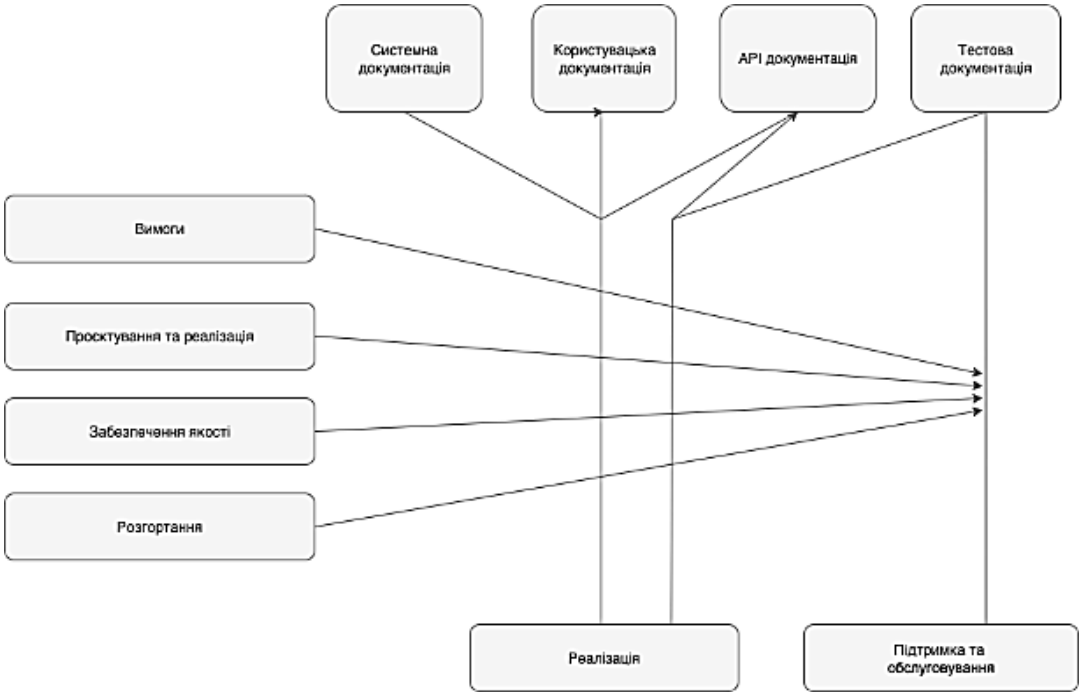

Рис.1. Компонентна діаграма категорій технічної документації

Таблиця 1

Типи документації згідно ISO/IEC/IEEE 26511–26515

Тип документа	Призначення	Створюється на етапі
Вимоги	Визначення цілей і функціоналу	t_1,t_2
Дизайн	Архітектура, компоненти, інтерфейси	t_3,t_4
Реалізація	Документація до коду, API	t_4
Тестування	Планування, звітування про якість	t_5
Експлуатація	Настанова користувача, адміністрування	t_5 і після релізу

Різнопланова технічна документація є ключовим фактором успішної реалізації ІТ проєкту, його ефективності та подальшої системної підтримки відповідного програмного продукту. Значна частина інформаційних систем із створення технічної документації в реальних ІТ проектах не відповідає критеріям актуальності, точності і підтримуваності. Для усунення таких недоліків необхідно використовувати стандартизовані інструменти і методики оцінки якості документації, створити інформаційну систему автоматичного генерування документів на основі коду та архітектури, а також інтеграція процедури документування у CI/CD-пайплайн. Якісна технічна документація − це не лише додатковий ресурс, а ефективна інвестиція у безпеку, підтримку, масштабованість і життєздатність ІТ проєкту та програмного продукту.

Етап формування обсягів роботи і розбивання на певні завдання (епіки)

Формування ІТ проєкту з розроблення програмного продукту формально можемо подати таким чином − P − програмний продукт; $T = \{t_1, t_2, .. t_n\}$ − множина етапів життєвого циклу проєкту; $E = \{e_1, e_2, ... e_k\}$ − множина епіків; $U = \{u_1, u_2, ... u_m\} \subseteq \bigcup E$ − множина користувацьких історій, що належать до тих чи інших епіків; $S = \{s_1, s_2, ... s_l\}$− множина безпекових шаблонів; C − програмний код; Q − якість продукту; τ_i− час виконання етапу t_i. Формально подамо процес інкрементного проекту із створення

програмного продукту у вигляді композиції функцій. Нехай I={I_1,…,I_r} − множина ітерацій (sprints): $P = \bigcup_{i=1}^{r} P_i$, де $P_i = (T_{test} \circ F \circ D \circ f_{sec})(U_i, S_i)$, при цьому P − повний програмний продукт сформований на основі r ітераційних частин, P_1,P_2,…,P_r; P_i − результати однієї і-тої ітерації (спринта); $\bigcup_{i=1}^{r} P_i$ − об'єднання всіх ітерацій, кожна з них формує певну частину продукту. Даний вираз подає одну ітерацію життєвого циклу розробки, а саме адаптацію вимог з урахуванням безпеки; проєктування (прототипування); кодування; тестування; результат − інкремент продукту P_i. Реалізовучи в сукупності таку послідовність ітерацій формується цільовий програмний продукт: $P = P_1 \cup P_2 \cup \ldots P_r$.

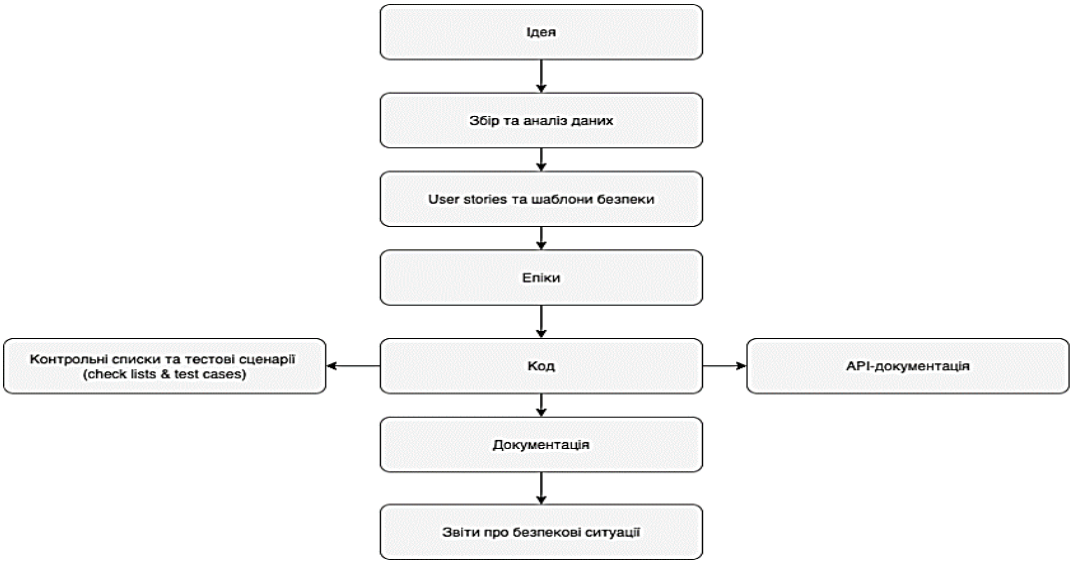

Рис.2. Алгоритм розроблення технічної документації IT проєкту

Це формульне подання логіки гнучкої (Agile) методології цільового розроблення програмного продукту, у якій кожна ітерація проходить повний цикл і вносить завершений функціональний фрагмент до кінцевого продукту. Динамічне оновлення документації відбувається на кожній з ітерацій, якщо проєкт реалізується за методологією Agile:

$$\forall I_k \in I,\ D_{tech}^{(k)} = D_{tech}^{(k-1)} \cup \Delta D_{tech}^{(k)}, \tag{5}$$

де I_k − k-та ітерація, I={I_1,I_2,…,I_r} − множина ітерацій або спринтів у проєкті, $\forall I_k \in I$ − для кожної ітерації I_k з множини I, $D_{tech}^{(k)}$ − стан технічної документації після завершення ітерації k і це є поданням всієї документації, актуальної на цей момент, $D_{tech}^{(k-1)}$ − стан документації на попередній ітерації (тобто на кроці k−1, $\Delta D_{tech}^{(k)}$ − множина нових або оновлених документів, що були додані в межах ітерації k; мова в цьому випадку йде про нові тест-кейси, уточнені вимоги, оновлені API тощо. На кожній ітерації життєвого циклу IT проєкту технічна документація на програмний продукт оновлюється. До наявного обсягу $D_{tech}^{(k-1)}$ додається нова частина $\Delta D_{tech}^{(k)}$, яка була створена або змінена в межах поточної ітерації. Документація в спринті k − це попередня документація плюс документні напрацювання, що були зроблені в межах цього спринта. Це формульне подання відображає процеси інкрементального ведення документації; підтримки в актуальному стані технічних артефактів (traceability, live docs); автоматизованого оновлення документації в конвеєрах CI/CD шляхом використання генераторів на кшталт Swagger, Sphinx; аудиту змін, що визначають, які частини документації оновлювалися і чому.

Наприклад:

На ітерації I_1 коли

$$D_{tech}^{(0)} = \emptyset, \text{ то } \Delta D_{tech}^{(1)} = \{VisionDocument, ProductBacklog\} \Rightarrow$$
$$D_{tech}^{(1)} = \emptyset \cup \{VisionDocument, ProductBacklog\}.$$

На ітерації I_2

$$D_{tech}^{(1)} = \{VisionDocument, ProductBacklog\}, \text{то } \Delta D_{tech}^{(2)} = \{Wireframes, APISpec\} \Rightarrow$$
$$D_{tech}^{(2)} = \{VisionDocument, ProductBacklog, Wireframes, APISpec\}.$$

На основі користувацьких історій та безпекових шаблонів, розробляються перші епіки (epic) для проєкту, які наповнюються завданнями безпосередньо в процесі реалізації, і на основі епіків проводиться перше наближене оцінювання тривалості проєкту:

$$T_{total} = \sum_{i=1}^{n} \tau_i,$$

де T_{total} – загальна тривалість проєкту, яка є сумою оцінених тривалостей усіх етапів або завдань, сформованих на основі епіків (epic) і користувацьких історій. Це перше наближене значення, яке надалі може уточнюватися; $t_i \in$T– етапи t_1 – формування епік та користувацьких історій; t_2 – застосування безпекових шаблонів; t_3– прототипування; t_4 – написання коду; t_5 – тестування першої версії продукту; n – кількість завдань (user stories) або підзадач, що входять до складу епіків на момент початкової оцінки. Кількість завдань визначається виходячи з декомпозиції епіків і безпекових шаблонів (security patterns); τ_i– орієнтовна тривалість виконання *i*-го завдання, виражена у вибраній одиниці часу (наприклад, людино-годинах, днях чи спринтах). Кожному етапу t_i відповідає своя тривалість τ_i, наприклад $t_1 \leftrightarrow \tau_1$; $t_2 \leftrightarrow \tau_2$; $t_n \leftrightarrow \tau_n$ Це значення зазвичай отримується на основі досвіду команди, історичних даних або методів оцінювання (Planning Poker, T-shirt sizing, Expert Judgment тощо).

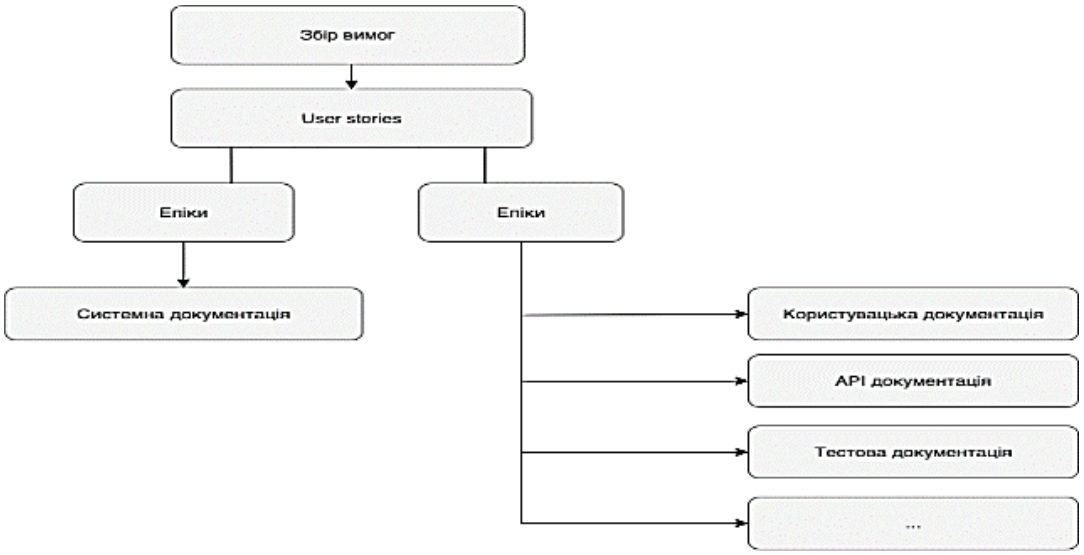

Рис.3 Етапи формування технічної документації IT проєкту

Таким чином, формула означає, що для початкової оцінки часу реалізації проєкту потрібно просумувати тривалості всіх завдань, визначених на основі користувацьких історій та безпекових шаблонів. На Рис. 3. зображено процес переходу від збору вимог до формування технічної документації шляхом аналізу користувацьких історій та інтеграційного їх подання у відповідних епіках. Алгоритм формування технічної документації IT проєкту грунтується на логічній послідовності процесів таких як збір вимог → формування користувацьких історій → створення епіків → технічна документація як консолідована система документів. Це композиція функцій відображає процес розроблення у межах однієї ітерації. Кожен епік $e_i \in E$ складається з набору користувацьких історій:

$$\forall e_i \in \text{E}, \exists U_i \subseteq \text{U}: e_i = \{u_{i_1}, u_{i_2}, \dots u_{i_{n_i}}\},$$

де e_i – окремий епік (epic), тобто велике завдання або функціональний блок проєкту; E– множина всіх епіків у системі; $\forall e_i \in$E – для кожного епіка e_i, що належить множині E; U– множина всіх користувацьких історій проєкту; $U_i \subseteq$U – існує підмножина U_i user stories, яка належить загальній множині U; $e_i = \{u_{i_1}, u_{i_2}, \dots u_{i_{n_i}}\}$ – епік e_i складається з набору user stories $u_{i_1}, u_{i_2}, \dots u_{i_{n_i}}$, які утворюють підмножину U_i .

Кожен епік e_i є множиною з кількох користувацьких історій, які належать до узагальненого переліку всіх користувацьких історій U. Тобто епік це група пов'язаних користувацьких історій, об'єднаних за функціональною або бізнес-логікою. Користувацькі історії є основою для формування документації в контексті наборів вимог, що формують первинне уявлення про функціональні очікування користувачів. На їх основі у технічній документації фіксуються набори початкових функціональних вимог, які пізніше уточнюються через епіки та задачі. Безпекові шаблони включаються до технічної документації як нефункціональні вимоги до програмного продукту, що забезпечують створення безпечного продукту. Для кожної користувацької історії $u_j \in U$ визначається відповідний безпековий шаблон:

$$s_j \in S: \forall u_j \in U, \exists s_j \in S: s_j \rightarrow u_j,$$

де u_j − окрема користувацька історія, що подана як опис окремої функціональної потреби користувача; U − множина всіх користувацьких історій IT проєкту; s_j − окремий безпековий шаблон, який реалізує типову вимогу безпеки; S − множина всіх безпекових шаблонів, доступних у системі; $s_j \rightarrow u_j$ − позначає, що шаблон безпеки s_j прив'язано або застосовано до користувацьких історій u_j ; $\forall u_j \in U$ − для кожної користувацької історії з множини U; $\exists s_j \in S$ − існує такий шаблон безпеки s_j, що застосовується до цієї історії. Для кожної користувацької історії в системі повинен бути призначений один (або більше) безпековий шаблон, який враховує або описує вимоги безпеки, пов'язані з реалізацією цієї історії. Це дозволяє вбудовувати вимоги безпеки на рівні планування функціоналу. Наприклад користувач бажає змінити пароль: u_j − «Користувач може змінити свій пароль»; s_j − шаблон безпеки з OWASP ASVS: «Secure password reset process», тобто ця функція має бути реалізована згідно вимог до процесу безпечної зміни пароля, що зафіксовано у шаблоні s_j. Це означає, що кожна користувацька історія модифікується з врахуванням вимог безпеки:

$$U_i^* = f_{sec}(U_i, S_i),$$

де (U_i, S_i)− вхідні дані для і-ї ітерації; U_i − набір користувацьких історій; S_i− відповідні безпекові шаблони; $f_{sec}(U_i, S_i)$ − функція інтеграції вимог безпеки, що формує безпеково адаптовані користувацькі історії. Таким чином, ми можемо коректно задавати стандарти безпеки системи. Кожен епік узагальнює групу користувацьких історій та дозволяє формувати структуру технічної документації по архітектурі програмного продукту, визначаючи компоненти, їх взаємодії, залежності та сценарії використання. Після створення епіків, саме в технічній документації подаються описи, які компоненти повинні бути реалізовані, які інтеграції будуть потрібні, і які ресурси будуть залучені, проводитиметься оцінювання обсягів робіт. Це формує основу для проведення попереднього оцінювання часу та вартості. У проектах, що реалізуються з використанням методологічних підходів, таких як Agile, DevOps, технічна документація формується поступово ітеративним наближенням. Саме при переході від користувацьких історій до епіків і задач − технічна документація починає набувати структурованого подання, що відповідає фазі мінімально необхідної документації. Етап формування епіків на основі користувацьких історій є критично важливим тригером для структурованого документування. Він дозволяє визначити, що саме буде реалізовано, якими силами, у які терміни, і водночас − закладає підґрунтя для написання архітектурних, функціональних і безпекових розділів технічної документації. Кожна користувацька історія або епік завершується реалізацією нового або зміненого функціоналу. На цьому етапі уже існує певна реалізація функціоналу у вигляді коду; є зв'язок між вимогами і реалізацією; виконано тестування, або принаймні написано тести; відомі всі зміни, які потрібно відобразити в документах. Дослідження автоматизованого документування після завершення кожної користувацької історії або епіку гарантує актуальність та повноту технічної документації, забезпечує трасування змін до джерел вимог, дозволяє спростити онбординг нових членів команди, сприяє реалізації процесів аудиту, підтримки та масштабування системи.

Таблиця 2

Компонент документації, формування яких можна автоматизувати на цьому етапі виконання ІТ проєкту.

Компонент документації	Що саме генерується або оновлюється автоматично
API-документація	Опис endpoint'ів, методів, параметрів, схем (OpenAPI, Swagger, JSDoc)
Технічна документація коду	Документація до класів, методів, інтерфейсів, автокоментарі (Javadoc, Docstrings)
Архітектурні схеми	Генерація UML/ER-діаграм на основі анотацій, моделей, PlantUML
Матриця відстежуваності вимог	Автоматичне оновлення матриці зв'язків: користувацька історія ↔ код ↔ тести ↔ документи
Текстові повідомлення	Звіти про результати тестування (Allure, JUnit XML)
Звіти про покриття тестами	Показники покриття коду тестами (JaCoCo)

Це до певної міри є критичним елементом інженерної культури в умовах реалізації методології Agile + DevOps.

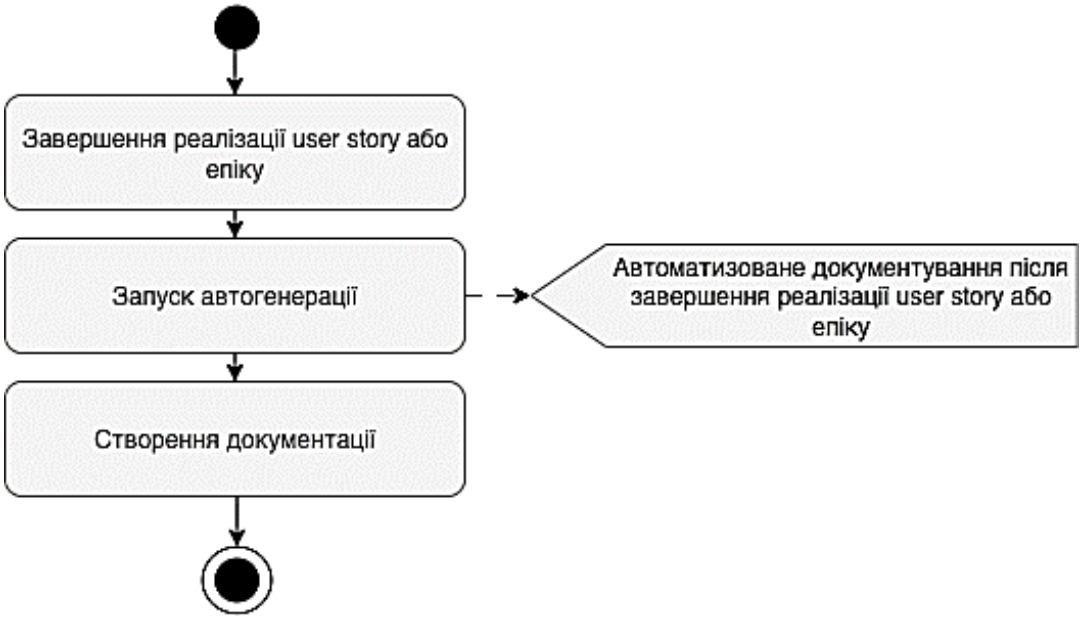

Рис. 4. Діаграма діяльності автоматизованого формування технічної документації ІТ проєкту на етапі епіків

Переваги такого підходу полягають у тому, що технічна документація оновлюється своєчасно - одразу після реалізації функціоналу, що забезпечує її актуальність та усуває розрив між реалізованим кодом і зафіксованими специфікаціями, що гарантує трасованість та дозволяє чітко простежити, які документи пов'язані з конкретними користувацькими історіями. Автоматизований режим зменшує обсяги ручної роботи, оскільки не потрібно вручну оновлювати API-опис, тестову документацію чи архітектурні діаграми. Уся документація формується за єдиними шаблонами, що сприяє її стандартизації та узгодженості процесів технічного супроводу ІТ проекту.

Етап інтеграції безпекових шаблонів

На етапі інтеграції безпечних шаблонів формуються документи, які забезпечують реалізацію, супровід і перевірку безпекових рішень у межах ІТ-проєкту. Основним документом є каталог безпечних шаблонів, де описано шаблони проєктування з прикладами коду, UML-діаграмами та сценаріями використання. Кожен шаблон вирішує конкретну проблему безпеки, має опис призначення, типовий контекст застосування та посилання на відповідні стандарти, як-от OWASP або NIST. Специфікація інтеграції визначає, як і де в системі застосовуються шаблони − на рівні API, бази даних або інтерфейсу

користувача − і фіксує вимоги до середовища, точки інтеграції та ролі відповідальних. Формуються також плани тестування і тест-кейси для перевірки реалізації шаблонів, зокрема тестування шифрування, автентифікації або захисту від атак. Матриця простежуваності допомагає відстежити відповідність між вимогами безпеки та реалізованими тестами. Політики безпеки формалізують обов'язкові вимоги до зберігання даних, доступу до сервісів, шифрування та опрацювання помилок. Крім того, створюються інструкції розробника, у яких подано покрокові настанови щодо впровадження шаблонів, README-файли, приклади виклику API або конфігурацій у CI/CD. Автоматизовані конфігурації у форматі YAML або JSON використовуються для перевірки наявності шаблонів у коді, зокрема через інструменти статичного аналізу. Генеруються звіти відповідності, які підтверджують, що код відповідає вимогам безпеки. У сукупності ці документи дозволяють реалізувати інтеграцію безпечних шаблонів уніфіковано, масштабовано й із можливістю постійного контролю в процесі розроблення.

Етап прототипування

Наступним етапом реалізації IT проєкту є етап прототипування, під час якого команда формує вайєр-фрейми користувацького інтерфейсу програмного продукту, що сприяє формулюванню цілісних системних вимог до функціоналу, з розподілом на модулі та завдання, а також початкове розуміння взаємозалежностей між різними частинами програмного продукту. Прототипування − це міст між концептуальними користувацькими історіями та технічною реалізацією. Саме на цьому етапі технічна документація вперше отримує чіткі візуальні і функціональні орієнтири, оскільки з'являється конкретика, уточнюється логіка взаємодій, структурується інтерфейс, і формуються технічні специфікації модулів. Прототип інформаційної системи фомально подамо:

$$\mathrm{P}t = F(U^*),$$

де Pt − прототип системи − результат виконання функції D, тобто узагальнене представлення (UI-макет, вайєрфрейм, модель екранів тощо), що візуалізує або формалізує майбутню поведінку програмного продукту; F − функція побудови прототипу − операція, що на основі вхідного набору користувацьких історій формує прототип інтерфейсу/модуля системи; U^*=$\{u_1^*, u_2^*, \dots\}$− множина всіх безпечно адаптованих користувацьких історій, які використовуються для створення прототипу; u_i^* − користувацька історія, яка модифікована з урахуванням вимог безпеки. Вона створена шляхом застосування відповідного безпекового шаблону до початкової користувацької історії u_i. Прототип системи Pt створюється за допомогою функції D, яка приймає на вхід множину користувацьких історій, модифікованих відповідно до вимог безпеки. Ці користувацькі історії вже враховують шаблони безпеки, тому побудований прототип реалізує не тільки функціональні вимоги, а й нефункціональні (зокрема безпекові) аспекти. До прикладу маємо користувацьку історію, в якій «Користувач вводить email для реєстрації». Формується безпековий шаблон: «Валідація вхідних даних + захист від SQL-ін'єкції», адаптовану користувацьку історію можемо подати так «Користувач вводить email, який валідується регулярним виразом; вхід захищено від SQL-ін'єкції». Ця користувацька історія увійде до U^*. Функція D сформує макет форми реєстрації з перевіркою введення Pt − графічного або логічного прототипу реєстраційного інтерфейсу з врахуванням вимог безпеки. Створення технічної документації IT проєкту на етапі прототипування полягає в уточненні та деталізації вимог, сприяє визначенню структури функціональних модулів, слугує візуальним джерелом процесів наповнення слотів фрейму розділів технічної документації, що є критичним для створення специфікацій інтерфейсів і взаємодій. Це дозволяє технічній команді зафіксувати вимоги до функціоналу, які раніше були описані словесно, включити ці уточнені вимоги до функціонального розділу технічної документації, перетворити графічні прототипи у специфікації UI-поведінки. У технічній документації IT проєкту формується розділ "Архітектура системи" з переліком модулів, їх зон відповідальностей та інтерфейсів. Вайєр-фрейми як технічна документація на інтерфейси описує API-запити до бекенду, які відповідають діям користувача на UI, уточнюють, яка саме інформація зберігається чи передається. Це відображається у розділах "API-документація" або "Вимоги до UI". Візуалізація процесів та процедур забезпечує покращення комунікації, використовуючи якісну документацію, оскільки Wireframes додаються до технічної документації як візуальні ілюстрації сценаріїв використання, полегшуючи

розуміння функціоналу для тестувальників, дизайнерів, аналітиків і кінцевих користувачів. В процесі реалізації ІТ проекту важливим джерелом формування технічної документації є програмний код, функцію його реалізації можемо подати:

$$H = M(\mathrm{Pt}, U^*),$$

де М − процес розроблення програмного забезпечення на основі прототипу та специфікацій користувацьких історій. Автоматичне створення документації із специфікацій інтерфейсів відбувається шляхом використання інструментів прототипування, які дозволяють експортувати структуру інтерфейсу у форматах JSON, HTML або Markdown. Вони зчитують назви екранів, взаємодії, зв'язки між кнопками і діями, а також дають змогу застосовувати плагіни для автогенерації документів. У результаті автоматично формується документ з описом екранів, полів, кнопок і переходів, що зберігається як складова технічної документації ІТ проєкту.

Таблиця 3

Перелік інструментів автоматичної генерації технічної документації для ІТ проєктів

Завдання	Інструменти
Генерація UI-специфікації	Figma → Figma2Spec, HTML2MD, Zeplin
Інтеграція з беклогом	Figma → Jira/Notion/ClickUp
Генерація API-документації	Stoplight, SwaggerHub, Insomnia, Postman
Трасування UI до функціоналу	Figma + Traceable.ai, PlantUML, Mermaid
Документація з JSON/GraphQL схем	GraphQL Voyager, JSON Schema Docs Generator

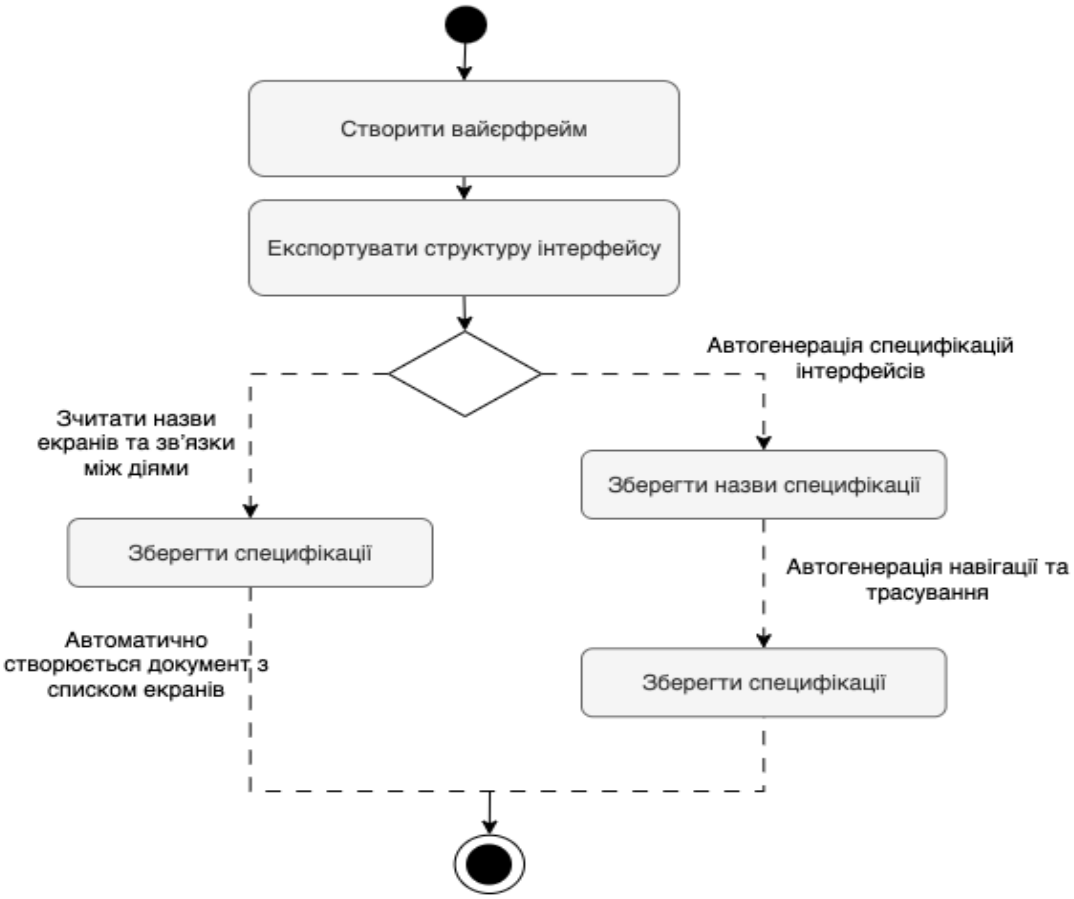

Рис. 5. Діаграма діяльності створення технічної документації ІТ проєкту на етапі прототипування

Етап реалізації програмного продукту

Наступний етап – реалізація. Якщо дотримуватись рекомендацій із написання програмного коду, слід враховувати його відповідність певним характеристикам, з яких в контексті документації найбільш важливими є коментарі до методів, класів, інтерфейсів, а також одна з вимог принципів SOLID, а саме single-responsibility, імплементація якої мала б створити передумови для успішного і ефективного автоматизованого тестування в межах проекту. Взаємозв'язок між формуванням технічної документації проєкту та етапом реалізації, в якому джерелом генерації документів виступає програмний код, полягає в тому, що вихідний код є основою для автоматизованого формування точнішої та підтримуваної документації, яка коректно та повно відображає реалізовану функціональність системи.

Етап тестування

Розглянемо ключові аспекти взаємозв'язків етапу тестування та процесів формування технічної документації. Коментарі присутні в коді є інформаційним джерелом для процесів автогенерації документів. Коментарі до методів, класів, інтерфейсів служать основою для генерації API-документації; інструкцій зі встановлення/виклику функцій; довідкових файлів у форматах HTML, Markdown, PDF. Такий підхід дозволяє знизити навантаження в режимі ручного документування та зменшити ризик розбіжностей між реалізацією та сформованими документами на ІТ проєкт. Реалізація принципу Single Responsibility (SRP) із SOLID означає, що кожен клас/метод виконує одну логічну функцію і забезпечується структуруванням документів. Це дозволяє створити логічно відокремлені блоки документації, прив'язані до окремих компонентів; упорядкувати технічну документацію за модулями; спрощує оновлення, оскільки, при зміні коду змінюється лише відповідний розділ документації. Автоматизоване тестування забезпечується документуванням тестів.

Рис.6. Діаграма діяльності формування технічної документації на основі коду

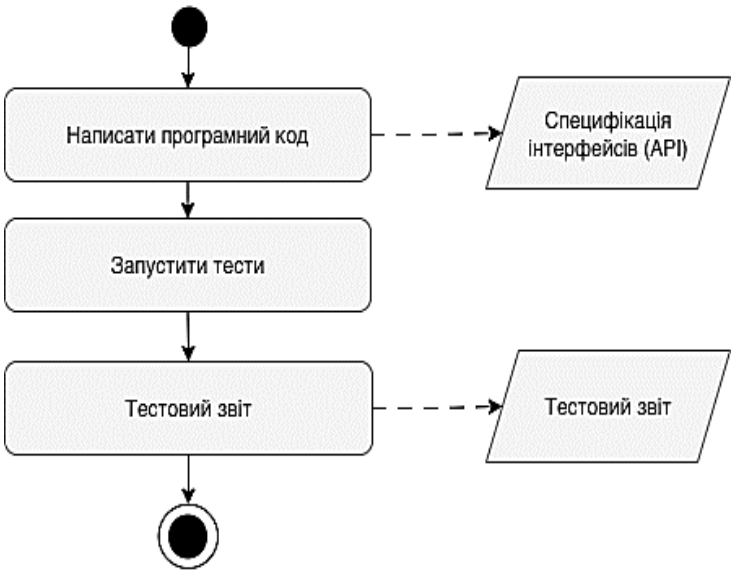

Рис.7. Діаграма діяльності ІС із формування документів на етапі тестування

Структурований, відповідно до SRP, код дозволяє легко створювати модульні тести, які служать джерелом прикладів виклику методів для документації; генерують автоматичні звіти (наприклад, coverage, CI/CD test reports), що входять у тестову документацію; підтверджують відповідність реалізації до задекларованих специфікацій. CI/CD забезпечує живе оновлення документації. Автоматизована система збирання (GitHub Actions, GitLab CI, Jenkins) може перегенеровувати документацію з коду на кожному коміті; публікувати її на внутрішньому порталі/сайті або на GitHub Pages; забезпечувати створення документів, синхронізоване з версією коду. Коментарі у коді як документоване знання забезпечують формування частини "живої документації", що виконує роль бази знань для нових учасників

проєкту; джерела даних для генерації технічних статей (з шаблонів); мапи зв'язків у складних архітектурах (на основі анотацій або метаданих). Програмний код є не лише об'єктом реалізації, а і джерелом документації. Якщо код написано структуровано (SRP) і прокоментовано відповідно до стандартів, це забезпечує автоматизовану побудову документації; її відповідність до фактичної реалізації; підтримку тестування, CI/CD і масштабування знань у команді.

За наявності доступу до репозиторію з прокоментованим кодом, а також при забезеченні прив'язки комітів до епіків, завдань та користувацьких історій, створюються передумови для автоматичного генерування checklists і test-cases як для ручного, так і для автоматизованого тестування. Цю частину технічної документації необхідно постійно підтримувати та оновлювати в процесі розробки програмного продукту. Вона забезпечує швидке залучення нових учасників команди до проєкту, а також дає змогу ефективно аналізувати стабільність програмного забезпечення, оцінювати його вразливість до атак та ризики витоку даних. Наявність повноцінної тестової документації сприяє аналізу покриття системи тестами (за допомогою матриці простежуваності вимог і тестів), а також дозволяє якісно оцінити необхідні ресурси для проведення регресивного тестування та інших видів верифікації. На основі сучасних систем керування версіями створюється репозиторій коду із можливістю інтеграції з інформаційною системою генерації технічної документації. До нього надається доступ інформаційній системі, яка автоматично формує супровідну проєктну документацію (архітектурну, тестову, експлуатаційну) на основі структури коду, метаданих і коментарів. Така інтеграція дозволяє підтримувати документацію в актуальному стані, забезпечує прозорість розробки та прискорює адаптацію нових учасників проєкту. За наявності доступу до репозиторію з документованим та структурованим кодом, включаючи коментарі до функцій і класів, README-файли, прив'язки комітів до епіків, задач і користувацьких історій, створюються умови для автоматизованого формування checklists і test-cases для ручного та автоматизованого тестування. Повноцінна документація для тестування, яка згенерована на основі коду з репозиторію і доповнена командою проекту, сприяє відслідковуванню взаємозалежності між модулями продукту, що дозволяє фільтрувати і застосовувати лише необхідний перелік тестових сценаріїв, необхідних для ручного тестування певного функціоналу, пропускаючи частини проекту, на які зміни не мали жодного впливу. Функція тестування T_{test} повертає показник покриття:

$$Q = T_{test}(\text{H}),$$

де Q − показник рівня покриття є числовим значенням (зазвичай у діапазоні Q∈ [0, 1], (0 − дана характеристика відсутня, 1 − повне покриття вимог) або [0 %, 100 %]), яке відображає рівень перевірки коду тестами. Чим вище значення показника, тим більшу частину коду охоплено тестуванням; H − програмний код, який є результатом реалізації функціональних і нефункціональних вимог. Це може бути код окремого модуля, компонента або всієї системи, T_{test} −функція тестування розглядається як процес або автоматизована система, яка виконує тести над кодом H. Вона може включати запуск юніт-тестів, інтеграційних тестів, функціональних/автоматизованих сценаріїв, статичний аналіз. Функція T_{test} застосовується до коду H і повертає значення Q, яке характеризує, наскільки повно протестовано реалізований код. Досвід практиків свідчить, що типові значення Q коливаються в межах Q=0.0, коли тестове покриття відсутнє (жодна частина коду не протестована), Q=0.75 в ситуації, коли 75% коду перевірено тестами; Q=1.0 у разі повного покриття, коли усі гілки, функції, класи протестовано. T_{test} враховує кількість протестованих рядків коду, покриття гілок (if/else), покриття шляхів виконання, кількість та якість тест-кейсів, наявність assert-операцій, логування результатів. Якщо команда розробила нову функцію в модулі авторизації, і юніт-тести перевіряють 90 % гілок цього коду, тоді: $T_{test}(\text{Cauth})$= Q=0.9. У конвеєрі CI/CD значення Q часто є умовою допуску до релізу якщо Q<0.8, збірка не допускається до production, якщо Q≥0.9, код вважається стабільним для інтеграції. Автоматизація процесів створення документації на етапі тестування є важливим інструментом забезпечення контролю якості, трасування дефектів і підтримки актуальності технічної документації. Вона ґрунтується на використанні автоматизованих фреймворків тестування у поєднанні з CI/CD-пайплайнами, які дозволяють автоматично генерувати звіти, метрики та специфікації.

Таблиця 4

Компоненти документів, що які генеруються автоматично

Компонент документації	Що генерується автоматично
Тестові звіти (Test Reports)	HTML, XML, JSON-звіти з результатами виконання тестів
Метрики покриття (Coverage)	Відсоток протестованого коду: рядки, гілки, функції
Звіт про дефекти (Defect Log)	Виявлені помилки, із зазначенням сценарію та коду
Traceability Matrix	Відповідність user stories ↔ тест-кейси ↔ реалізація
Результати статичного аналізу	Попередження про стиль, складність, вразливості

Автоматизоване документування на етапі тестування забезпечує низку важливих переваг. По-перше, це актуальність, оскільки документація створюється одночасно з виконанням тестів, що гарантує її відповідність поточному стану системи. По-друге, забезпечується повнота, завдяки трасуванню жодна користувацька історія не залишається без покриття. Крім того, автоматизація значно підвищує швидкість − немає потреби вручну описувати результати тестування. Документація інтегрується безпосередньо у процес розробки, що робить її природною частиною CI/CD-пайплайну. Нарешті, створюються зручні умови для аудиту та аналітики: стає легко оцінити якість, стабільність та готовність продукту до релізу. Подамо програмний продукт як результат послідовного застосування кількох функцій/етапів до вхідних даних:

Рис. 8. Діаграма діяльності автоматизованого формування документів на етапі тестування

$$\text{Ui та Si: } P_i = (T_{test} \circ \text{Ff} \circ \text{D} \circ f_{\text{sec}})(U_i, S_i),$$

де P_i − вихідний результат створення для і-го модуля програмного продукту; U_i − набір користувацьких історій або вимог користувача для і-го модуля чи функціоналу програмного продукту, S_i− відповідні безпечні шаблони; f_{sec} − функція безпеки, яка застосовується першою, додаючи безпекові вимоги, зокрема шаблони безпеки, політики доступу, контроль ризиків; D − документи (технічні специфікації, архітектурні діаграми, API-контракти), перетворення узгоджених вимог (з урахуванням безпеки) у структуровані специфікації; Ff – формалізовані документи і моделі, готові для використання у CI/CD, тобто створення моделі, використання трасування, забезпечення однозначності опису; T_{test} − процес автоматизованого тестування на основі формалізованих документів та моделей автоматично створюються тестові сценарії, звіти, контроль узгодженості між кодом і документацією. Це остання трансформація перед отриманням підсумкового артефакту P_i.

Таким чином, сформовано формальне подання, яке описує конвеєр процесів автоматизації документації та контролю якості, де на вхід подаються вимоги та специфікація, вони доповнюються безпековими вимогами, формалізуються, документуються і проходять автоматичну перевірку. На рис 9. наведено приклад інтеграційної карти викликів API, щоб продемонструвати логіку взаємодії між фронт-та бекендом, з викликами, параметрами, порядком. Сформована таким чином документація дає змогу правильно спроектувати і запустити необхідні інтеграційні тести, для перевірки сумісності фронт- та бекендом частин проекту і їх коректного функціонування.

User Story	Епік	Test	Коміт	Документація
Користувач може реєструватися в системі	Реєстрація	Test_001	e1a145d	Розділ 1
Користувач може входити до системи	Реєстрація	Test_002	5216h3d	Розділ 2
Користувач може скинути пароль	Реєстрація	Test_003	4f3e223	Розділ 2
Користувач може скинути пароль	Реєстрація	Test_001	4f3e223	Розділ 2
	Реєстрація	Test_003		

Рис. 9 Матриця взаємозв'язків користувацьких історій, епіків, тестів, комітів та документації

В поданій нижче матриці продемонстровано, як окремі користувацькі історії пов'язані з епіками, тестами, комітами у системі контролю версій та відповідними розділами документації.

Таблиця 5

Перелік документів, які формуються на кожному етапі проєкту

Етап t_i	Назва етапу	Формалізація	Документи A(ti)
t_1	Етап формування обсягів роботи і розбивання на певні завдання (епіки)	E={e_i}, U=Ue_i	Візуалізований документ, вимоги зацікавлених сторін, модель варіантів використання, беклог продукту
t_2	Інтеграція безпечних шаблонів	$u_j^* = f_{sec}(u_j, s_j)$,	Специфікація вимог безпеки, модель загроз, план захисту даних
t_3	Прототипування	$\mathrm{P}t = F(U^*)$,	Дизайн UI/UX, специфікація прототипу, вайрфрейми, діаграма блок-схеми користувача
t_4	Розробка програмного коду	$H = M(Pt, U^*)$	Документ архітектури програмного забезпечення (SAD), Специфікація проектування модулів, Документація коду, Довідник API
t_5	Тестування	$\mathrm{Q} = T_{test}(H)$	План тестування, тестові випадки, звіт про тестування, звіт про помилку
$\sum$	Підсумок реалізації повного життєвого циклу	$P_i = (T_{test} \circ \mathrm{Ff} \circ \mathrm{D} \circ f_{\mathrm{sec}})(U_i, S_i)$,	Технічний посібник, Посібник з розгортання, Посібник користувача, Документ з технічного обслуговування

Такий підхід до створення документації сприяє її постійному оновленню і актуалізації в процесі розроблення, оскільки більшість документів матиме безпосередню прив'язку до частин коду в репозиторії, відповідно зміни в цих частинах повинні запускати автоматичне оновлення документації. На рис. 10 показано інтерфейси редагування (ліворуч) та перегляду (праворуч) сторінки документації у внутрішній wiki-платформі. Для створення повноцінної консолідованої документації на ІТ проєкт необхідно описувати архітектуру, інфраструктуру проекту, формувати опис процесу оновлення проекту, правил внесення змін. Окрім технічної документації проекту, важливим аспектом розроблення безпекових систем, є коректна генерація звітів щодо безпекових ситуацій, що виникають в процесі роботи системи.

Такі звіти повинні містити повний опис залучених і ідентифікованих учасників безпекової ситуації, хронологічну послідовність подій, дані, надані інтелектуальними агентами, і рішення прийняте системою. Вони дозволять проводити глибший аналіз інцидентів, впроваджувати нові безпекові сценарії, а також забезпечувати документування інцидентів з правоохоронними органами.

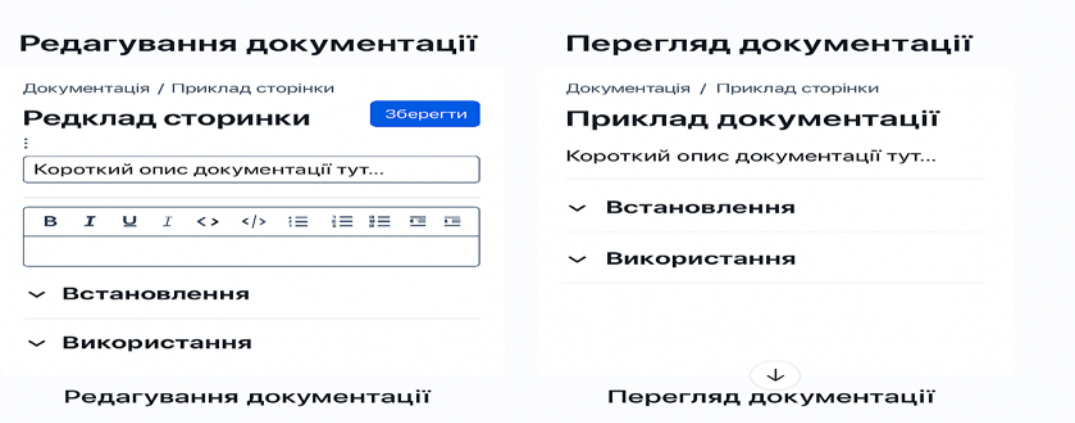

Рис.10. Інтерфейс редагування та перегляд документації

У процесі реалізації IT проєкту із створення продукту технічна документація формується відповідно до специфіки кожного типу застосунку або інформаційної системи, оскільки забезпечує повноцінне розуміння структури, функціонування та інтеграційних можливостей продукту як для розробників, так і для користувачів або технічних фахівців. Наприклад, під час розроблення системи моніторингу безпеки для багатоквартирного житлового комплексу необхідно підготувати детальний опис інфраструктури - серверної архітектури, камер спостереження та мережевих протоколів взаємодії.

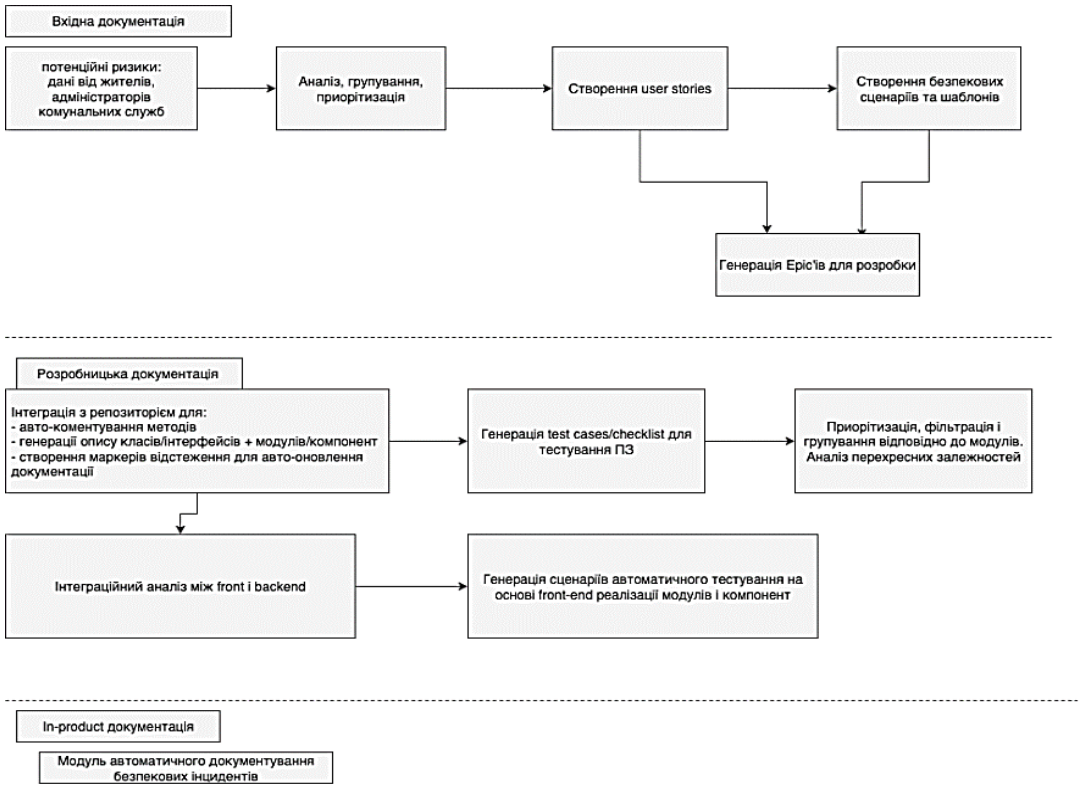

Рис.11. Етапи розроблення технічної документації

Документація повинна також містити інтерфейси програмного забезпечення (API), які забезпечують зв'язок із мобільним застосунком мешканців. Важливим елементом є документи з описом сценаріїв реагування на типові загрози, зокрема ситуації вторгнення або порушення периметру, що охороняється, а також алгоритми автоматизованого реагування на ці загрози. Окрім того, обов'язковими є розділи з інструкціями для обслуговуючого персоналу, адміністраторів та технічних фахівців щодо користування інформаційною системою. Такий підхід також підтримує безперервну інтеграцію й гнучке оновлення систем

при поєднанні DevOps- та Agile-методологій. Таке поєднання передбачає гнучке планування, інкрементальну реалізацію функціоналу (Agile), паралельно із впровадженням автоматизованих процесів тестування, CI/CD-конвеєрів, моніторингу та розгортання (DevOps). У результаті забезпечується цілісний підхід до управління життєвим циклом ПЗ − від ідеї повноти функціоналу до етапу експлуатації, з акцентом на швидкість поставки, надійність релізів та безперервне вдосконалення продукту.

Висновки

Запропоновано формалізовану модель побудови технічної документації, що описана через математичні функції, множини та композиції. Модель дозволяє структурувати документи за етапами життєвого циклу, оцінювати повноту покриття та забезпечувати відповідність сучасним стандартам (IEEE 830, ISO/IEC 26514). Встановлено, що застосування безпекових шаблонів на рівні user stories є критично важливим для інтеграції нефункціональних вимог у документацію, особливо для безпеково чутливих систем. Показано, що використання інструментів автогенерації, коментарів до коду, CI/CD-конвеєрів та трасувальних матриць дозволяє мінімізувати ручну працю, підвищити актуальність і зменшити ризики втрати знань. Було також описано роль документації у відображенні покриття тестами, оновленні технічної інформації на кожній ітерації, а також у зв'язку між функціональністю, архітектурою та API. У результаті дослідження було сформульовано змістовні, структурні, організаційні та технологічні вимоги до технічної документації: повнота, актуальність, точність, трасованість, доступність і відповідність стандартам. Розроблено тривимірну модель технічного документу, що визначається його типом, етапом, на якому створений або оновлений, відповідно до обраної методології. Таким чином, технічна документація у сучасних ІТ-проєктах є основою для керованого розвитку, безпечної реалізації та якісного супроводу програмних систем. Її формалізоване створення та автоматизація в умовах CI/CD забезпечують сталість і масштабованість програмного продукту на всіх етапах його життєвого циклу.

СПИСОК ЛІТЕРАТУРИ

Aimar, A. (2001). *Introduction to software documentation* (Technical report). CERN, IPT Group, IT Division. https://cds.cern.ch/record/572356

Behutiye, W., Rodríguez, P., Oivo, M., Aaramaa, S., Partanen, J., & Abhervé, A. (2022). Towards optimal quality requirement documentation in agile software development: A multiple case study. *Journal of Systems and Software*, 183, 111112. https://doi.org/10.1016/j.jss.2021.111112

Blome, M. (2025, January 29). Automation in technical documentation: Faster, smarter, better. *Adoc Studio Blog*. https://www.adocstudio.app/blog/automation-in-technical-documentation

Brockmann, R. J. (1990). *Writing better computer user documentation*: From paper to hypertext. John Wiley & Sons.

Buchgeher, G., Klammer, C., Dorninger, B., & Kern, A. (2018). Providing technical software documentation as a service: An industrial experience report. *In Proceedings of the Asia-Pacific Software Engineering Conference (APSEC 2018)* (pp. 581–590). IEEE. https://doi.org/10.1109/APSEC.2018.00073

IEEE Standards Association. (2020). *IEEE Std 1063-2020: IEEE standard for software user documentation. Institute of Electrical and Electronics Engineers*. https://doi.org/10.1109/IEEESTD.2020.9055273

Kolomiiets, I. (2024). *Mastering technical documentation in DevOps.* Attract Group. https://attractgroup.com/blog/mastering-technical-documentation-in-devops/

Low, R., & Ford, H. (1995). *Writing user documentation: A practical guide for those who want to be read. Prentice Hall*.

Rai, S., Belwal, R. C., & Gupta, A. (2022). A review on source code documentation. ACM Transactions on *Management Information Systems*, 13(5). https://doi.org/10.1145/3519312

Ramesh, B., Cao, L., & Baskerville, R. (2010). Agile requirements engineering practices and challenges: An empirical study. *Information Systems Journal*, 20(5), 449–480. https://doi.org/10.1111/j.1365-2575.2007.00259.x

Ries, R. (1990). IEEE standard for software user documentation. *In Proceedings of the International Conference on Professional Communication* (pp. 66–68). IEEE. https://doi.org/10.1109/IPCC.1990.111154

Synko, A., & Peleshchyshyn, A. (2020). Software development documenting – Documentation types and standards. *Scientific Journal of the Ternopil National Technical University*, 98(2), 120–128. https://doi.org/10.33108/visnyk-tntu2020.02

FORMULATION OF TECHNICAL DOCUMENTATION FOR SOFTWARE

Oleg Zakhariya, Nataliia Kunanets, Yuriy Zhovnir

Lviv Polytechnic National University,
Information Systems and Networks Department, Lviv, Ukraine
E-mail: ozakhar@gmail.com, ORCID: 0009-0008-6979-129X
E-mail: nek.lviv@gmail.com, ORCID: 0000-0003-3007-2462
E -mail: zhovnir@astra.in.ua, ORCID: 0009-0006-6186-2861

Summary. The article explores the methodology for creating technical documentation in software development projects, taking into account the principles of Agile and DevOps. It proposes formal approaches for building a document system as an integrated part of the IT project life cycle. The study examines the stages of transforming user stories into structured documents, the role of security templates, prototyping, and source code as inputs for generating specifications. Emphasis is placed on automated documentation, including auto-generation of API documentation, test reports, and the use of traceability matrices. The relationship between documents and project stages (epics, commits, tests) within the CI/CD pipeline is analyzed. The paper also presents formalisms and a mathematical model, diagrams, and architectural approaches that ensure the completeness, relevance, and structured nature of technical documentation.

Keywords: technical documentation, software, user stories, epics, security templates, prototyping, formalization, CI/CD, Agile, DevOps, API documentation, requirements traceability, auto-generated documentation, code-based documentation.