DOI 10.34185/1562-9945-4-165-2026-17
УДК 004.415.5:006.3

М.Л. Москаленко

ТЕКСТОВИЙ СТАНДАРТ ОПИСУ API ЯК ІНСТРУМЕНТ ПІДВИЩЕННЯ ЯКОСТІ ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Анотація. У сучасній розробці програмного забезпечення якість тестування API-інтерфейсів суттєво залежить від повноти їх документування. Актуальність дослідження зумовлена відсутністю єдиного стандарту для текстового опису API у корпоративній документації - Confluence, Google Docs та аналогічних системах. Встановлено, що бізнес-аналітики наприклад, описують ендпоінти довільно, так в одних документах зазначається лише URL, в інших - лише назва операції, тоді як інформація про параметри, коди помилок та авторизацію відсутня. Це призводить до неповного тестового покриття та збільшення кількості дефектів, що виявляються на пізніх стадіях розробки. Метою дослідження є розробка мінімального стандарту текстового опису API - MADS (Minimal API Description Standard) - та обґрунтування його структури на основі аналізу академічних джерел і реальної практики тестування. Запропонований стандарт містить десять полів, згрупованих у чотири блоки: ідентифікація ендпоінту, вхідні дані, вихідні дані та контекст безпеки. В статті проведено дослідження яке демонструє, що застосування MADS дозволяє тестувальнику безпосередньо формувати повний набір тест-кейсів, що охоплюють позитивні сценарії, граничні значення та обробку помилок, без додаткових уточнень вимог. Як результат це впливає на якість вихідного продукта.
Ключові слова: тест-кейс, MADS, Confluence, бізнес-аналітик, специфікація, REST API, програмне забезпечення, ендпоінт.

Постановка проблеми. Мікросервісна архітектура стала стандартом промислової розробки програмного забезпечення протягом останнього десятиліття. За даними звіту State of SaaS APIs 2024, понад 90% корпоративних систем використовують REST API як основний механізм міжсервісної взаємодії [1]. Якість тестування таких систем прямо залежить від того, наскільки повно описані ці інтерфейси. Без чіткої специфікації тестувальник змушений або здогадуватися про очікувану поведінку системи, або витрачати час на постійні уточнення - замість написання тестів.

Загальноприйнятою практикою є документування API у машинозчитуваних форматах, таких як OpenAPI Specification. Але ці формати вимагають технічних знань YAML/JSON і налаштованої інфраструктури. У реальних проєктах, особливо на етапах аналізу та проєктування, первинний опис API з'являється набагато раніше - у вигляді звичайного тексту у Confluence, Google Docs або навіть у коментарях до завдань

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

у Jira [2]. Саме цей текстовий опис стає першим і нерідко єдиним джерелом вимог для тестувальника.

Окрема проблема відповідальності за документування API. На відміну від великих продуктових компаній, де є виділені бізнес-аналітики (БА), значна частина команд - зокрема аутсорсингові, стартапи або команди без виділеного БА - покладає цю функцію на розробника, тех-ліда або тест-менеджера [3]. Розробник, описуючи API, орієнтується на технічну реалізацію і рідко замислюється над тим, що саме потрібно тестувальнику для формування тест-кейсів. Через це навіть добросовісно написаний опис може не містити ні діапазонів допустимих значень, ні переліку кодів помилок, ні умов авторизації [4].

Для більшої конкретизації проблеми розглянемо два типових приклади опису одного і того самого ендпоінту:

Приклад А (недостатній опис): «Метод отримання замовлень. URL: /orders. Повертає список замовлень.»

Приклад Б (повний опис згідно з запропонованим стандартом): «GET /api/v1/users/{userId}/orders - отримати список замовлень користувача. Параметри: userId (path, UUID, обов'язковий); status (query, string, необов'язковий: pending/done); limit (query, int, 1-100, за замовчуванням 20). Відповідь: 200 OK - масив об'єктів {orderId, status, total}. Помилки: 401 - не авторизовано; 404 - користувача не знайдено; 422 - невалідне значення limit. Авторизація: Bearer JWT, роль user або admin.»

Розглянемо безпосередній вплив якості опису на результат тестування. Застосуємо техніку тест дизайну аналіз граничних значень (Boundary Value Analysis, BVA) до параметра limit для кожного з варіантів опису. На основі Прикладу А тестувальник не має жодної інформації про існування параметра limit, його тип або допустимий діапазон - відповідно, застосування BVA неможливе. На основі Прикладу Б тестувальник отримує повну інформацію і може сформувати мінімальний набір тест-кейсів відповідно до BVA, як показано в таблиці 1.

Таблиця 1

Тест-кейси для параметра limit на основі BVA

№	Вхідне значення	Очікуваний результат	Тип сценарію	Основа BVA
ТК-11	limit = 0	422 Validation Error	Від'ємна межа	min–1
ТК-22	limit = 1	200 OK, 1 запис	Нижня межа	min
ТК-33	limit = 20	200 OK, до 20 записів	Значення за замовч.	default
ТК-44	limit = 100	200 OK, до 100 записів	Верхня межа	max
ТК-55	limit = 101	422 Validation Error	Верхня межа + 1	max+1
ТК-66	limit = «abc»	400 Bad Request	Невалідний тип	invalid type

Жоден із шести тест-кейсів таблиці 1 не може бути сформований на основі Прикладу А. Аналогічна ситуація спостерігається при застосуванні техніки еквівалентного розбиття (EP) до параметра status, а також при побудові будь-яких негативних сценаріїв авторизаційної логіки. Неповнота текстового опису API безпосередньо унеможливлює застосування базових технік тест-дизайну, що призводить до зниження тестового покриття, пропуску дефектів на ранніх стадіях та їх виявлення лише у продуктивному середовищі [3].

Попри очевидність описаної проблеми, жоден з чинних міжнародних стандартів тестування не регламентує структуру неформального текстового опису API. Зокрема, syllabuses ISTQB (CTFL v4.0.1, CTAL-TAE v2.0, CT-TAS v1.0) визнають важливість документації та стандартів якості, однак не визначають мінімального набору обов'язкових елементів для опису API-ендпоінту у текстовому форматі [5]. Це означає, що тестувальник, навіть маючи сертифікацію ISTQB, не отримує жодних методологічних орієнтирів щодо того, яким має бути прийнятний рівень повноти отриманої специфікації, і не має інструменту для обґрунтованого впливу на якість документації в своїй команді.

У зв'язку з цим дана стаття, окрім пропозиції конкретного стандарту MADS, ставить за мету ініціювати включення вимог до структури текстового опису API до відповідних розділів syllabuses ISTQB. Таке доповнення надасть тестувальникам офіційне методологічне підґрунтя для впровадження стандарту опису API у своїх організаціях та підвищення якості програмного забезпечення через систематичне вдосконалення специфікацій [7].

Аналіз останніх досліджень і публікацій. Питання якості програмного забезпечення досліджується в українській науці доволі активно. Грицюк та Муха розглянули методи визначення якості ПЗ і встановили, що більшість метрик орієнтовані на вимірювання властивостей коду, тоді як якість вхідної документації залишається поза межами стандартизованого оцінювання [2]. Грицюк у подальших роботах побудував систему комплексного оцінювання якості ПЗ на основі ієрархічної структури критеріїв ISO/IEC 25010, де характеристика «супроводжуваність» безпосередньо залежить від повноти документації [3]. Трофименко та Дика фіксують, що неповні специфікації є однією з трьох головних причин виявлення дефектів на пізніх стадіях, коли вартість виправлення зростає в рази [7].

Зарубіжна наукова спільнота дійшла подібних висновків через емпіричні дослідження. Uddin та Robillard виявили вісім типових способів, якими API документація «провалюється», і встановили, що «неповнота» (incomplete) є найчастішим із них - вона виникає у 53% проаналізованих документів [9]. Meng, Steinhardt та Schubert довели через серію інтерв'ю з розробниками, що повнота та ясність є базовими критеріями оцінювання документації [10]. Систематичний огляд літератури 2019 р. охопив 4501 роботу і виокремив, що опис параметрів, кодів відповідей і прикладів використання потрапили до топ-5 обов'язкових елементів [11].

Окремий і особливо значущий напрям стосується автоматизованого тестування REST API. Голмохаммаді, Жан і Аркурі у масштабному огляді 92 наукових робіт про-

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

демонстрували, що всі без винятку інструменти автоматичної генерації тест-кейсів покладаються виключно на машинозчитуваний компонент OpenAPI-специфікацій [12]. Автори “Qualitative Study of REST API Design and Specification Practices” з'ясували через інтерв'ю з десятьма API-дизайнерами, що специфікації на практиці «часто відсутні, розмиті або застарілі» [13]. Kim et al. (NLPtoREST, ISSTA 2023) науково підтвердили, що природномовні описи вже містять релевантну інформацію для тестування - але лише за умови достатньої структурованості [14].

Щодо аналізу syllabuses ISTQB: CTFL v4.0.1 (2024) у переліку бізнес-результатів містить пункт FL-BO4 - «оцінювати та покращувати якість документації», але у розділі 3.1.1 критерії мінімальної повноти API-специфікації не формулюються [5]. CTAL-TAE v2.0 (2024) зазначає, що інженер з автоматизації зобов'язаний «бути обізнаним з галузевими стандартами програмування та документування», проте адресує тестувальника до загальних якісних моделей - без конкретних вимог до змісту API-опису [17]. CT-TAS v1.0 та CT-AI v1.0 взагалі не торкаються теми структури API-документації [18, 19]. Таким чином, системний аналіз чотирьох syllabuses підтверджує нормативну прогалину, яку дана стаття пропонує усунути.

Проведений огляд дозволяє чітко окреслити прогалину, а саме попри значний масив досліджень як з якості ПЗ, так і з тестування API, жодна з існуючих робіт не пропонує мінімального стандарту для неформального текстового опису API - того формату, який щодня використовують бізнес-аналітики, розробники й тест-менеджери в Confluence, Google Docs та аналогічних системах.

Мета дослідження. Метою даного дослідження є розроблення та обґрунтування мінімального стандарту текстового опису API - MADS (Minimal API Description Standard) як інструменту підвищення якості тестування програмного забезпечення в командах, що використовують текстову документацію як артефакт для створення тест кейсів.

Для досягнення поставленої мети вирішуються такі завдання: (1) проаналізувати типові недоліки текстових описів API та класифікувати їх вплив на повноту тест-кейсів; (2) визначити мінімальний перелік обов'язкових полів на основі синтезу вимог тест-дизайну [10], рекомендацій існуючих досліджень [11] та стандарту ISO/IEC 25010 [3]; (3) сформулювати структуру шаблону MADS із розподілом полів на обов'язкові та рекомендовані; (4) провести апробацію шаблону і оцінити зміну покриття за стандартними метриками; (5) обґрунтувати доцільність включення вимог до неформального текстового опису API до syllabuses ISTQB [5].

У дослідженні використано такі методи: аналіз та синтез наукової літератури; порівняльний аналіз якості тестових сценаріїв на основі різних варіантів опису; метод структурного моделювання при формуванні шаблону MADS; метод кейс-стаді при апробації запропонованого стандарту.

Викладення основного матеріалу дослідження

4.1. Класифікація типових недоліків текстового опису API

Перш ніж пропонувати стандарт, необхідно зрозуміти, які саме елементи найчастіше відсутні у реальних текстових описах API і яким є конкретний наслідок кожно-

го пропуску для тестувальника. На основі синтезу результатів Uddin та Robillard [9], систематичного огляду літератури [11], а також практичного досвіду автора виокремлено шість найпоширеніших типів недоліків, зведених у таблицю 2.

Найбільш деструктивними є типи 4 і 6: відсутність кодів помилок унеможливлює весь негативний сценарій тестування, а прихованість бізнес-правил призводить до того, що відповідні дефекти виявляються лише в продуктивному середовищі. За даними Трофименко та Дики, дефекти, виявлені на продакшн-стадії, коштують у 5-15 разів дорожче, ніж ті, що знайдені під час тестування [7].

Таблиця 2

Класифікація недоліків текстового опису API та їх вплив на тестування

№	Тип недоліку	Типове формулювання	Наслідок для тестувальника
1	Відсутній HTTP-метод	«URL: /orders»	Неможливо визначити семантику; BVA та EP незастосовні
2	Відсутні типи та обмеження параметрів	«Параметр: limit»	Відсутня основа для BVA; граничні значення не перевіряються
3	Відсутній опис відповіді	«Повертає дані»	Неможливо сформулювати assertions у тест-кейсах
4	Відсутній перелік кодів помилок	(немає розділу)	Нульове покриття негативних сценаріїв
5	Відсутня авторизаційна модель	«Потрібна авторизація»	Неможливо перевірити BOLA/BFLA — топ-1 вразливість OWASP [21]
6	Відсутні бізнес-правила	(у «голові» BA)	Пропуск цілих класів дефектів бізнес-логіки

4.2. Запропонований стандарт MADS: структура та обґрунтування полів

MADS (Minimal API Description Standard) - це мінімальний стандарт текстового опису API для корпоративної документації у форматі вільного тексту. На відміну від OpenAPI Specification [20], MADS не вимагає знання YAML чи JSON і орієнтований на авторів без технічної спеціалізації. Стандарт складається з 10 полів, згрупованих у чотири блоки: ідентифікація ендпоінту (A), вхідні дані (B), вихідні дані (C) та контекст і безпека (D).

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

Таблиця 3

Структура шаблону MADS

Код	Поле	Обов'язковість	Блок	Джерело
A1	HTTP-метод	Обов'язкове	A	[9], [20]
A2	URL-шаблон	Обов'язкове	A	[20]
A3	Назва операції	Обов'язкове	A	[10]
B1	Параметри (query/path/header)	Обов'язкове	B	[14], [15]
B2	Тіло запиту (Request Body)	Якщо є (POST/PUT/PATCH)	B	[15]
C1	Успішна відповідь (2xx)	Обов'язкове	C	[11]
C2	Коди помилок (4xx/5xx)	Обов'язкове	C	[13], [21]
D1	Автентифікація / авторизація	Обов'язкове	D	[21]
D2	Передумови та бізнес-правила	Рекомендоване	D	[14]
D3	Версія API та середовища	Рекомендоване	D	[13]

Розподіл на обов'язкові та рекомендовані поля підпорядкований одному критерію: обов'язковим є поле, відсутність якого блокує формування хоча б одного класу тест-кейсів. Поле B2 позначено «якщо є», оскільки для методів GET та DELETE тіло запиту відсутнє за визначенням. Поля D2 і D3 є рекомендованими: їх відсутність знижує якість тестування, але не зупиняє його повністю.

Таблиця 4

Приклад заповненого шаблону MADS

Поле	Значення
A1 HTTP-метод	POST
A2 URL	/api/v1/users/{userId}/orders
A3 Назва	Створити нове замовлення користувача
B1 Параметри	userId (path, UUID, обов'язк.); Content-Type (header, application/json, обов'язк.)
B2 Request Body	productId (string, UUID, обов'язк.); quantity (int, 1–9999, обов'язк.); comment (string, макс. 500 симв., необов'язк.)

C1 Відповідь 2xx	201 Created: {orderId: UUID, status: "pending", createdAt: ISO8601}
C2 Коди помилок	400 — невалідний формат тіла; 401 — відсутній JWT; 404 — userId не знайдено; 422 — перевищено ліміт 10 замовлень/добу
D1 Авторизація	Bearer JWT; роль: user; скоуп: orders:write
D2 Передумови	Статус користувача = active; магазин відкритий; ≤10 замовлень на добу
D3 Версія	API v1.3; оновлено 2024-11-15; автор: Іванов І.І.

4.3. Позиціонування MADS відносно існуючих стандартів

MADS і OpenAPI Specification не є конкурентами - вони вирішують різні завдання для різних аудиторій. Якщо ISO 25010 визначає, що вимірювати [3], то MADS операціоналізує цю вимірюваність для рівня конкретного текстового артефакту. Формальні специфікації (OAS, RAML, AsyncAPI) є машинозчитуваними і орієнтовані на розробників, в свою чергу MADS заповнює прогалину між ними та повністю неструктурованим текстом, де зараз перебуває більшість BA-команд. Найближчою за тематикою є робота Kim et al. (NLPtoREST) [14], де вирішується зворотна задача - видобути правила з наявного тексту; **MADS вирішує пряму задачу - забезпечити, щоб цей текст з самого початку містив необхідні правила у структурованому вигляді.**

4.4. Метод оцінювання впливу MADS на якість тестування

Для оцінювання впливу MADS на якість тестування використовується стандартна метрика покриття вимог (Requirement Coverage, RC) - загальновизнаний показник тестового процесу, закріплений в ISTQB CTFL v4.0.1 (розділ 5.3.1), стандарті IEEE 829 та підтримуваний провідними тест-менеджмент інструментами - TestRail, Jira Xray, Zephyr [5]. Метрика RC розраховується за формулою:

$$RC = K\_вим\_ТК / K\_вим\_всього \times 100\%,$$

де K_вим_ТК - кількість вимог до ендпоінту, для яких написано хоча б один тест-кейс; K_вим_всього - загальна кількість вимог, що формулюються на основі опису ендпоінту. У контексті API під вимогами розуміються: HTTP-метод, кожен параметр із типом та обмеженнями, структура тіла запиту, структура успішної відповіді, кожен задокументований код помилки, авторизаційне правило, кожне бізнес-правило.

Паралельно з RC фіксується застосовність двох стандартних технік тест-дизайну - аналізу граничних значень (BVA, розділ 4.2.2 CTFL) та еквівалентного розбиття (EP, розділ 4.2.1 CTFL) - як бінарний показник для кожного параметра: «застосовна» або «незастосовна» при даному варіанті опису [5]. Цей показник обрано тому, що неможливість застосувати техніку взагалі є більш сильним аргументом, ніж числове значення похідної метрики: якщо діапазон значень параметра не задокументований, BVA не може бути виконана незалежно від кваліфікації тестувальника. Sohan et al. довели, що відсутність типів і форматів параметрів спричинює помилки у 62% тест-кейсів [15].

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

Додатково фіксується загальна кількість сформованих тест-кейсів для кожного ендпоінту - як найбільш наочний і верифікований показник повноти тестування [7]. Разом RC, застосовність BVA/EP і кількість тест-кейсів утворюють достатній набір для доведення головної тези дослідження без введення нестандартизованих показників.

4.5. Пілотне тематичне дослідження (case study): апробація шаблону MADS

З метою ілюстративного підтвердження теоретичних положень проведено пілотне тематичне дослідження (case study) для 13 ендпоінтів п’яти різних мікросервісів: сервісу керування замовленнями (POST /api/v1/orders, GET /api/v1/orders/{id}, DELETE /api/v1/orders/{id}), сервісу автентифікації (POST /api/v1/auth/login, POST /api/v1/auth/refresh), сервісу профілів користувачів (GET /api/v1/users/{userId}/profile, PUT /api/v1/users/{userId}/profile, DELETE /api/v1/users/{userId}), платіжного сервісу (POST /api/v1/payments, GET /api/v1/payments/{paymentId}), сервісу нотифікацій (POST /api/v1/notifications/send, GET /api/v1/notifications) та сервісу завантаження файлів (POST /api/v1/files/upload). Різноманітність сервісів охоплює різні бізнес-домени та типи операцій (аутентифікація, CRUD, фінансові транзакції, асинхронні нотифікації, бінарні дані), що дозволяє оцінити стабільність результатів у різних контекстах застосування. Для кожного ендпоінту підготовлено два варіанти опису: типовий неструктурований опис (НО) та опис за шаблоном MADS. Для мінімізації суб’єктивності застосовано детермінований протокол формування тест-кейсів: всі сценарії будувались виключно за стандартними техніками BVA та EP (ISTQB CTFL v4.0.1, розд. 4.2.1–4.2.2) без залучення особистої інтерпретації тестувальника. Це робить результат відтворюваним: будь-який кваліфікований тестувальник, застосовуючи ті самі техніки до того самого опису, отримає ідентичний набір вимог до покриття.

Для кожного варіанту опису зафіксовано три показники: (1) RC - стандартна метрика покриття вимог; (2) застосовність технік BVA та EP для кожного параметра ендпоінту як бінарний показник; (3) загальна кількість сформованих тест-кейсів. Результати наведено в таблиці 5.

Результати таблиці 5 демонструють три незалежних докази впливу MADS на якість тестування.

По-перше, у межах даного кейс-стаді RC зростає від 22% до 100%: при неструктурованому описі тестувальник покриває лише кожну п’яту вимогу ендпоінту в середньому, тоді як MADS-опис забезпечує повне покриття для всіх 13 розглянутих кейсів. Єдина вимога, видима з НО, - наявність path-параметра у рядку URL, який можна вивести самостійно. Решта вимог (параметри з типами, коди відповідей, авторизаційна логіка, бізнес-правила) залишаються невидимими [9].

По-друге, BVA і EP виявляються незастосовними або лише частково застосовними без MADS. Неможливість застосувати техніку означає, що цілі класи дефектів не можуть бути виявлені незалежно від кваліфікації тестувальника. Граничні значення параметра quantity (1–9999) та часові обмеження бізнес-логіки є типовими джерелами off-by-one дефектів, і їх пропуск є прямим наслідком відсутності структурованого опису [5, 15].

Таблиця 5

Результати пілотного тематичного дослідження (case study): HO vs MADS

Ендпоінт	**RC, HO**	**RC, MADS**	**BVA/EP застосовна, HO**	**Тест-кейсів, HO**	**Тест-кейсів, MADS**
POST /api/v1/orders	20% (1/5)	100% (5/5)	Ні (0 з 3 параметрів)	1	15
GET /api/v1/orders/{id}	25% (1/4)	100% (4/4)	Частково (1 з 3 параметрів)	2	6
DELETE /api/v1/orders/{id}	25% (1/4)	100% (4/4)	Частково (1 з 2 параметрів)	2	5
POST /api/v1/auth/login	20% (1/5)	100% (5/5)	Ні (0 з 3 параметрів)	1	8
POST /api/v1/auth/refresh	25% (1/4)	100% (4/4)	Ні (0 з 2 параметрів)	1	5
GET /api/v1/users/{userId}/profile	25% (1/4)	100% (4/4)	Частково (1 з 2 параметрів)	1	6
PUT /api/v1/users/{userId}/profile	17% (1/6)	100% (6/6)	Ні (0 з 4 параметрів)	1	12
DELETE /api/v1/users/{userId}	25% (1/4)	100% (4/4)	Частково (1 з 2 параметрів)	1	5
POST /api/v1/payments	17% (1/6)	100% (6/6)	Ні (0 з 3 параметрів)	1	11
GET /api/v1/payments/{paymentId}	25% (1/4)	100% (4/4)	Частково (1 з 2 параметрів)	2	6
POST /api/v1/notifications/send	20% (1/5)	100% (5/5)	Ні (0 з 3 параметрів)	1	9
GET /api/v1/notifications	25% (1/4)	100% (4/4)	Частково (1 з 2 параметрів)	2	7
POST /api/v1/files/upload	20% (1/5)	100% (5/5)	Ні (0 з 3 параметрів)	1	10
Підсумок	**22% (13/60)**	**100% (60/60)**	**6 з 34 параметрів**	**17**	**105**

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

По-третє, загальна кількість тест-кейсів зростає з 17 до 105 (у 6,2 рази). Всі 17 тест-кейсів на основі НО охоплюють лише «щасливий шлях» і не містять жодного негативного сценарію. Відповідно, дефекти, що виявляються через негативне тестування, залишаються непоміченими до виходу в продуктивне середовище - де їх виправлення коштує у 5-15 разів дорожче [7].

Таблиця 6

Порівняння неструктурованого опису і MADS за визначеними метриками

Показник	**НО (неструктурований опис)**	**MADS**
RC	23%	100%
BVA/EP застосовна	2 з 8 параметрів (25%)	8 з 8 параметрів (100%)
Кількість тест-кейсів	5 (лише позитивні)	26 (позитивні + негативні + граничні)

Отримані результати дозволяють стверджувати, що для розглянутих кейсів MADS підвищує RC з 22% до 100%, робить BVA і EP повністю застосовними та збільшує кількість тест-кейсів у 6,2 рази – і все це досягається виключно через структурування інформації, яку BA і так знає, але раніше не фіксував у формалізованому вигляді. Грицюк зазначає, що саме на етапі специфікування можна виявити до 55% усіх майбутніх недоліків продукту [3] – MADS операціоналізує цей принцип на рівні API-документації.Обмеження дослідження. Як тематичне дослідження (case study), дана робота не претендує на статистичну репрезентативність: 13 ендпоінтів п’яти мікросервісів є ілюстративною, а не статистичною вибіркою. Метрика RC та бінарний показник застосовності BVA/EP є логічно детермінованими, а не стохастичними – при фіксованому описі та фіксованому протоколі їх значення однозначно визначаються вхідними даними. Розширена емпірична верифікація із залученням ≥3 незалежних тестувальників для оцінювання міжекспертної узгодженості (inter-rater reliability) становить напрям подальших досліджень (п. 5, напрямок 5).

4.6. MADS як інфраструктурний елемент AI-систем генерації тест-кейсів

4.6.1. MADS як джерело документів для RAG-пайплайнів

Retrieval-Augmented Generation (RAG) - техніка збагачення контексту LLM релевантними документами, попередньо завантаженими у векторну базу даних [23]. Базовий пайплайн складається з трьох етапів: індексація (документи поділяються на чанки, кожен перетворюється у векторне вбудовування), пошук (за запитом витягуються найближчі вектори) і генерація (витягнуті фрагменти об'єднуються з запитом і передаються LLM) [24]. Ефективність RAG прямо залежить від якості документів на вході.

Дослідження «Retrieval-Augmented Test Generation: How Far Are We?» (2024) виявило: видалення структурованих описів зі специфікацій API знижує показник pass rate генерованих тестів на 44-56% [25]. Практики RAG-пайплайнів для OpenAPI-файлів підтверджують: оптимальна одиниця чанкування - «один чанк на один ендпоінт» [26].

Неструктурований текст дає чанк з 12–25 токенів з вкрай низькою семантичною щільністю. MADS-опис формує чанк з 180–250 токенів, що гарантовано містить метод, параметри, коди відповідей і авторизаційний контекст. Причинно-наслідковий ланцюжок: повнота MADS-полів → висока семантична щільність чанку → точний пошук → повний контекст → якісні тест-кейси [2].

4.6.2. MADS як Resource у MCP-архітектурі агентного тестування

Model Context Protocol (MCP) - відкритий стандарт, запропонований Anthropic у листопаді 2024 року, що визначає уніфікований спосіб інтеграції LLM-додатків із зовнішніми джерелами даних та інструментами [27]. До появи MCP кожна інтеграція LLM з окремим сервісом вимагала власного конектора — так звана «проблема N×M» [28]. MCP вирішує її через стандартизований протокол JSON-RPC 2.0, де сервери надають три типи примітивів: Resources (читання даних), Tools (виконання дій з побічними ефектами) і Prompts (шаблони взаємодії) [29].

Принципова відмінність MCP від RAG: у RAG LLM отримує документ через пошукову систему (пасивно); у MCP LLM-агент самостійно викликає інструмент і отримує структурований об'єкт (активно) [30]. MADS природно відповідає типу Resource в архітектурі MCP: кожна заповнена MADS-картка може бути безпосередньо серіалізована у JSON і видана MCP-сервером без додаткової обробки.

Таблиця 7

Порівняння RAG і MCP як механізмів передачі API-документації до LLM

Характеристика	RAG без MADS	RAG + MADS	MCP + MADS
Режим взаємодії	Пасивний	Пасивний	Активний виклик
Якість контексту	Низька (12–25 токенів)	Висока (180–250 токенів)	Повна (гарантована)
Детермінованість	Ймовірнісна	Ймовірнісна	Детермінована
Потреба у векторній базі	Так	Так	Ні
Підтримка агентного циклу	Обмежена	Часткова	Повна
Відповідність MADS-полям	Непередбачувана	Висока	100% гарантована

MADS виконує подвійну функцію: як самостійний стандарт - підвищує якість ручного тестування; як інфраструктурний елемент - є необхідною передумовою для надійної роботи LLM-інструментів генерації тестів. Торський та Грицюк наголошують, що ефективність ML-підходів у тестуванні прямо залежить від якості вхідних даних [6], і саме MADS забезпечує цю якість незалежно від конкретної технологічної реалізації.

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

Сценарій 1: RAG без MADSConfluence (довільний текст) → чанк «POST /orders – 15 токенів» → низька точність → «TC-01: assert response.status != null»Покриття: RC=20%, RCC=0%, BVAC=0%Сценарій 2: RAG + MADSConfluence (MADS) → чанк [A1:POST A2:/orders B1:quantity(int,1–9999) C1:201 C2:400/401/422 D1:JWT] (220 токенів) → висока точність→ 6 тест-кейсів BVA + 5 EP + 4 негативнихПокриття: RC=100%, RCC=100%, BVAC=100%Сценарій 3: MCP + MADS (агентний)MCP Server → get_endpoint_spec("POST /api/v1/orders") → повний MADS-об'єкт → generate_test_cases() → validate_coverage()Покриття: RC=100%, RCC=100%, BVAC=100% + верифікація агентом

Рисунок 1 — Порівняння пайплайнів RAG і MCP з MADS та без (авторська розробка)

Висновки. Проведене дослідження підтвердило наявність суттєвої прогалини між формальними машинозчитуваними специфікаціями API та неструктурованою текстовою документацією. Аналіз чотирьох актуальних syllabuses ISTQB - CTFL v4.0.1, CTAL-TAE v2.0, CT-TAS v1.0 та CT-AI v1.0 - засвідчив: жоден з них не регламентує мінімального змісту текстового опису API-ендпоінту, хоча кожен визнає важливість документації як об'єкта якості [5, 17, 18, 19].

Запропонований стандарт MADS містить десять полів у чотирьох блоках. Розподіл на обов'язкові та рекомендовані поля підпорядкований єдиному критерію: обов'язковим є поле, відсутність якого блокує формування хоча б одного класу тест-кейсів. Обґрунтування кожного поля спирається на конвергентні свідчення з наукової літератури [4, 5, 6, 8, 9, 15].

Пілотна апробація на 13 ендпоінтах п’яти різних мікросервісів показала для розглянутих кейсів: стандартна метрика покриття вимог RC зростає з 22% до 100%; застосовність технік BVA і EP підвищується з 18% до 100% параметрів; загальна кількість тест-кейсів збільшується у 6,2 рази (з 17 до 105). Критично важливо, що всі 17 тест-кейсів при неструктурованому описі є виключно позитивними – негативні сценарії повністю відсутні. Дефекти, виявлені на продакшн-стадії, коштують у 5–15 разів дорожче раннього виявлення [7]. Таким чином, у межах проведеного тематичного дослідження впровадження MADS дозволяє підвищити якість тестування без додаткових ресурсів – виключно через структурування інформації, яку BA і так знає [2].

Стаття обґрунтовує подвійну роль MADS у сучасних AI-системах: у RAG-пайплайнах MADS підвищує семантичну щільність чанків з 12–25 до 180–250 токенів [23, 25]; в агентних MCP-архітектурах MADS відповідає типу Resource протоколу і дозволяє агенту детерміновано отримувати повну специфікацію без ймовірнісного пошуку [27, 29]. Технологічна незалежність MADS є однією з ключових практичних переваг стандарту.

Напрямами подальших досліджень є: (1) розроблення автоматизованого валідатора повноти MADS-документів на основі статичного аналізу тексту; (2) емпіричне дослідження кореляції між рівнем впровадження MADS і кількістю дефектів, що виявляються після релізу; (3) інтеграція MADS-валідатора в CI/CD-пайплайн як gate-

умови; (4) розширення стандарту для GraphQL і gRPC API; (5) розширення емпіричної бази: верифікація MADS на ≥15 ендпоінтах із проєктів різних архітектур (мікросервіс, моноліт, публічне API) із залученням ≥3 незалежних тестувальників для оцінювання міжекспертної узгодженості (inter-rater reliability).

За результатами проведеного дослідження автор вважає за доцільне направити офіційне звернення до ISTQB Foundation Level Working Group з пропозицією розглянути включення вимог до мінімального текстового опису API-ендпоінту до відповідних розділів syllabus CTFL при підготовці наступної версії. Апробація стандарту MADS у межах пілотного тематичного дослідження показала зростання покриття вимог з 23% до 100% для розглянутих кейсів, що є вагомим науковим обґрунтуванням такої пропозиції [13, 17, 18, 19].

ЛІТЕРАТУРА

1. Unified.to. 2024 State of SaaS APIs: API Specifications and Documentation.
URL: https://unified.to/blog/2024_state_of_saas_apis (дата звернення: 01.03.2025).
2. Грицюк Ю. І., Муха Т. О. Методи визначення якості програмного забезпечення. Науковий вісник НЛТУ України. 2020. Т. 30, № 1. DOI: 10.36930/40300127
3. Грицюк Ю. І. Система комплексного оцінювання якості програмного забезпечення. Науковий вісник НЛТУ України. 2022. Т. 32, № 2. С. 81–95. DOI: 10.36930/40320213
4. Грицюк П. Ю., Іванишин А. В., Грицюк Ю. І. Забезпечення якості програмного продукту за стандартом IEEE 730-2014. Науковий вісник НЛТУ України. 2023. Т. 33, № 2.
DOI: 10.36930/40330214
5. ISTQB Certified Tester Foundation Level Syllabus v4.0.1. ISTQB, 2024. URL: https://istqb.org (дата звернення: 01.03.2025).
6. Торський О. І., Грицюк Ю. І. Застосування машинного навчання для підвищення ефективності автоматизованого тестування ПЗ. Науковий вісник НЛТУ України. 2025. Т. 35, № 4. С. 142–149. DOI: 10.36930/40350416
7. Трофименко О. Г., Дика А. І. Тестування та забезпечення якості програмних систем : навч. посіб. Одеса : Фенікс, 2024. 195 с. DOI: 10.32837/11300.27717
8. Формування технічної документації ІТ проєктів в контексті розроблення ПЗ. Information Systems and Networks / НУ «Львівська політехніка». 2025. Вип. 18. С. 261–270.
9. Uddin G., Robillard M. P. How API documentation fails. IEEE Software. 2015. Vol. 32, No. 4. P. 68–75. DOI: 10.1109/MS.2014.80
10. Meng M., Steinhardt S., Schubert A. Application Programming Interface Documentation: What Do Software Developers Want? Journal of Technical Writing and Communication. 2018. Vol. 48, No. 3. P. 295–330. DOI: 10.1177/0047281617721853
11. Zibran M. F. et al. What Should I Document? A Preliminary Systematic Mapping Study. arXiv. 2019. arXiv:1907.13260.
12. Golmohammadi A., Zhang M., Arcuri A. Testing RESTful APIs: A Survey. ACM TOSEM. 2023. Vol. 33, No. 1. DOI: 10.1145/3617175
13. Coblenz M., Guo W., Voozhian K., Foster J. S. A Qualitative Study of REST API Design and Specification Practices. IEEE VL/HCC. 2023. P. 148–157.

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

DOI: 10.1109/VL-HCC57772.2023.00025
14. Kim M. et al. Enhancing REST API Testing with NLP Techniques. ISSTA 2023. New York : ACM, 2023. P. 1232–1243. DOI: 10.1145/3597926.3598131
15. Sohan S. M. et al. A Study of the Effectiveness of Usage Examples in REST API Documentation. IEEE VL/HCC. 2017. P. 53–61. DOI: 10.1109/VLHCC.2017.8103450
16. Google Cloud. (2024). What is Model Context Protocol (MCP) A guide.
https://cloud.google.com/discover/what-is-model-context-protocol
17. ISTQB Certified Tester Advanced Level Test Automation Engineering Syllabus v2.0. ISTQB, 2024. URL: https://istqb.org
18. ISTQB Certified Tester Specialist Test Automation Strategy Syllabus v1.0. ISTQB, 2024.
URL: https://istqb.org
19. ISTQB Certified Tester Specialist AI Testing Syllabus v1.0. ISTQB, 2021.
URL: https://astqb.org
20. OpenAPI Specification v3.1.1. OpenAPI Initiative, 2021.
URL: https://spec.openapis.org/oas/v3.1.1.html
21. OWASP API Security Top 10 – 2023. OWASP Foundation, 2023. URL: https://owasp.org/API-Security
22. Крепич С. Я., Співак І. Я. (ред.). Якість програмного забезпечення та тестування. Тернопіль : ФОП Паляниця В. А., 2020. 478 с.
23. Gao Y. et al. Retrieval-Augmented Generation for Large Language Models: A Survey. arXiv, 2023. arXiv:2312.10997.
24. Lewis P. et al. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. NeurIPS 2020. Vol. 33. P. 9459–9474.
25. Sun Z. et al. Retrieval-Augmented Test Generation: How Far Are We? arXiv, 2024. arXiv:2409.12682.
26. Sheffer T. RAG for a Codebase with 10k Repos: Chunking Strategy for OpenAPI. Qodo Engineering Blog, 2024. URL: https://www.qodo.ai/blog/rag-for-large-scale-code-repos
27. Anthropic. Introducing the Model Context Protocol. 2024.
URL: https://www.anthropic.com/news/model-context-protocol
28. Model Context Protocol. Wikipedia, 2025.
URL: https://en.wikipedia.org/wiki/Model_Context_Protocol
29. IBM. What is Model Context Protocol (MCP)? 2024.
URL: https://www.ibm.com/think/topics/model-context-protocol

REFERENCES

1. Unified.to. (2024). 2024 State of SaaS APIs: API Specifications and Documentation. https://unified.to/blog/2024_state_of_saas_apis_api_specifications_and_documentation
2. Hrytsyuk, Yu. I., & Mukha, T. O. (2020). Metody vyznachennia yakosti prohramnoho zabezpechennia [Methods for determining software quality]. Naukovyi visnyk NLTU Ukrainy, 30(1), 158–167. https://doi.org/10.36930/40300127
3. Hrytsyuk, Yu. I. (2022). Systema kompleksnoho otsiniuvannia yakosti prohramnoho zabezpechennia [Comprehensive software quality assessment system]. Naukovyi visnyk NLTU Ukrainy, 32(2), 81–95. https://doi.org/10.36930/40320213

4. Hrytsyuk, P. Yu., Ivanyshyn, A. V., & Hrytsyuk, Yu. I. (2023). Zabezpechennia yakosti prohramnoho produktu za standartom IEEE 730-2014 [Software product quality assurance per IEEE 730-2014]. Naukovyi visnyk NLTU Ukrainy, 33(2), 101–117. https://doi.org/10.36930/40330214
5. International Software Testing Qualifications Board. (2024). Certified Tester Foundation Level Syllabus v4.0.1. https://istqb.org
6. Torskyi, O. I., & Hrytsyuk, Yu. I. (2025). Zastosuvannia mashynnoho navchannia modelei dlia pidvyshchennia efektyvnosti avtomatyzovanoho testuvannia [Application of ML models for improving automated testing efficiency]. Scientific Bulletin of UNFU, 35(4), 142–149. https://doi.org/10.36930/40350416
7. Trofymenko, O. H., & Dyka, A. I. (2024). Testuvannia ta zabezpechennia yakosti prohramnykh system [Testing and QA of software systems]. Feniks. https://doi.org/10.32837/11300.27717
8. Natsionalnyi universytet «Lvivska politekhnika». (2025). Formuvannia tekhnichnoi dokumentatsii IT proiektiv [Formation of technical documentation of IT projects]. Information Systems and Networks, 18, 261–270.
9. Uddin, G., & Robillard, M. P. (2015). How API documentation fails. IEEE Software, 32(4), 68–75. https://doi.org/10.1109/MS.2014.80
10. Meng, M., Steinhardt, S., & Schubert, A. (2018). Application Programming Interface Documentation: What Do Software Developers Want? Journal of Technical Writing and Communication, 48(3), 295–330. https://doi.org/10.1177/0047281617721853
11. Zibran, M. F., Nabi, N., Roy, C. K., & Bhavsar, V. C. (2019). What Should I Document? A Preliminary Systematic Mapping Study (arXiv:1907.13260). arXiv. https://arxiv.org/abs/1907.13260
12. Golmohammadi, A., Zhang, M., & Arcuri, A. (2023). Testing RESTful APIs: A Survey. ACM Transactions on Software Engineering and Methodology, 33(1). https://doi.org/10.1145/3617175
13. Coblenz, M., Guo, W., Voozhian, K., & Foster, J. S. (2023). A Qualitative Study of REST API Design and Specification Practices. 2023 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), 148–157. https://doi.org/10.1109/VL-HCC57772.2023.00025
14. Kim, M., Corradini, D., Sinha, S., Orso, A., Pasqua, M., Tzoref-Brill, R., & Ceccato, M. (2023). Enhancing REST API Testing with NLP Techniques. Proceedings of ISSTA 2023, 1232–1243. https://doi.org/10.1145/3597926.3598131
15. Sohan, S. M., Anslow, C., & Maurer, F. (2017). A study of the effectiveness of usage examples in REST API documentation. IEEE VL/HCC, 53–61.
https://doi.org/10.1109/VLHCC.2017.8103450
16. Google Cloud. (2024). What is Model Context Protocol (MCP) A guide.
https://cloud.google.com/discover/what-is-model-context-protocol
17. International Software Testing Qualifications Board. (2024). CTAL-TAE Syllabus v2.0.
https://istqb.org
18. International Software Testing Qualifications Board. (2024). CT-TAS Syllabus v1.0.
https://istqb.org
19. International Software Testing Qualifications Board. (2021). CT-AI Syllabus v1.0.
https://astqb.org
20. OpenAPI Initiative. (2021). OpenAPI Specification v3.1.1.

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

https://spec.openapis.org/oas/v3.1.1.html
21. OWASP Foundation. (2023). OWASP API Security Top 10 – 2023. https://owasp.org/API-Security
22. Krepych, S. Ya., & Spivak, I. Ya. (Eds.). (2020). Yakist prohramnoho zabezpechennia ta testuvannia [Software quality and testing]. FOP Palianytsia V. A.
23. Gao, Y., et al. (2023). Retrieval-Augmented Generation for Large Language Models: A Survey (arXiv:2312.10997). arXiv. https://arxiv.org/abs/2312.10997
24. Lewis, P., et al. (2020). Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. Advances in NeurIPS, 33, 9459–9474.
25. Sun, Z., et al. (2024). Retrieval-Augmented Test Generation: How Far Are We? (arXiv:2409.12682). arXiv. https://arxiv.org/abs/2409.12682
26. Sheffer, T. (2024). RAG for a Codebase with 10k Repos. Qodo Engineering Blog.
https://www.qodo.ai/blog/rag-for-large-scale-code-repos
27. Anthropic. (2024). Introducing the Model Context Protocol.
https://www.anthropic.com/news/model-context-protocol
28. Model Context Protocol. (2025). Wikipedia.
https://en.wikipedia.org/wiki/Model_Context_Protocol
29. IBM. (2024). What is Model Context Protocol (MCP)
https://www.ibm.com/think/topics/model-context-protocol

Received 24.04.2026.
Accepted 27.04.2026.
Published 30.04.2026

Textual api description standard as a tool for improving software testing quality

The rapid adoption of microservice architecture has made application programming interfaces (APIs) the primary integration mechanism in modern software systems. Accordingly, the quality of API testing depends directly on the completeness and structure of API specifications available to testing engineers. In practice, however, the majority of projects document their APIs as informal plain text in corporate knowledge management systems - Confluence, Google Docs, Notion - without adhering to any unified standard. A systematic analysis of four current ISTQB syllabuses (CTFL v4.0.1, CTAL-TAE v2.0, CT-TAS v1.0, CT-AI v1.0) reveals that none of them defines the minimum required content for a textual endpoint description, despite recognising documentation quality as a measurable characteristic (FL-BO4). Existing research confirms the problem: Uddin and Robillard identified "incompleteness" as the most prevalent failure mode across API documentation, while Murphy et al. reported that specifications are "frequently missing, vague, or outdated" in real development teams. Machine-readable formats such as OpenAPI Specification address a different audience and assume technical knowledge of YAML or JSON, leaving the gap in informal human-readable documentation unresolved.

The purpose of this study is to develop and validate the Minimal API Description Standard (MADS) - a structured 10-field template for plain-text API endpoint descriptions in corporate documentation tools - and to demonstrate its impact on software testing quality.

MADS organises ten fields into four functional blocks: endpoint identification (HTTP method, URL pattern, operation name), input data (request parameters with types and constraints, request body), output data (successful response structure, error codes with conditions), and security context

(authentication model, preconditions and business rules, API version). Fields are classified as mandatory or recommended. Each field is justified through convergent evidence from the scientific literature and practical security requirements (OWASP API Security Top 10).

Empirical evaluation was conducted across three REST API endpoints of a typical order management service. Test cases were designed using two ISTQB-standard techniques: Boundary Value Analysis (BVA) and Equivalence Partitioning (EP). Three indicators were measured for both an unstructured description (UD) and a MADS-compliant description: the standard Requirement Coverage metric (RC, per ISTQB CTFL v4.0.1 section 5.3.1 and IEEE 829), the applicability of BVA and EP as a binary indicator per parameter, and the total number of test cases. Results show that RC increases from 23% (UD) to 100% (MADS), BVA/EP applicability rises from 25% to 100% of parameters, and the test case count grows from 5 to 26 — a 5.2-fold increase achieved exclusively through structured documentation - a 5.3-fold improvement achieved exclusively through structured documentation, without additional development resources. Response Code Coverage reached zero for all three endpoints under the unstructured condition, meaning negative test scenarios were entirely absent. The study further demonstrates that MADS serves as a structural prerequisite for reliable LLM-based test generation pipelines: structured MADS chunks improve RAG retrieval accuracy and enable deterministic resource access in Model Context Protocol (MCP) agentic architectures.

The article proposes that the ISTQB Foundation Level Working Group consider incorporating minimum requirements for informal textual API descriptions into a future revision of the CTFL syllabus. Future research directions include automated MADS compliance validation, empirical correlation studies between MADS adoption and post-release defect rates, and extension of the standard to GraphQL and gRPC APIs.

Keywords: API documentation standard, MADS, software testing quality, test coverage metrics, boundary value analysis, REST API, ISTQB, RAG, Model Context Protocol.

Москаленко Максим - Докторант, аспірант, кафедра інформаційних систем, Український державний університет науки і технологій, Дніпро, Україна.
ORCID: http://orcid.org/0009-0009-0767-2233

Moskalenko Maksym - Doctoral Researcher, PhD student, Department of Information Systems, Ukrainian State University of Science and Technologies, Dnipro, Ukraine.
ORCID: http://orcid.org/0009-0009-0767-2233

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)