

Теоретичні відомості 1. Дослідницьке тестування, оракули та схеми покриття

Що ви робите, коли досліджуєте продукт

Тестування — це процес оцінювання продукту через пізнання його шляхом переживання, дослідження та експериментування з ним.

Перевіряння (checking) — це механістичний процес верифікації тверджень про продукт.

Обидва визначення належать методології Rapid Software Testing (RST) Джеймса Баха та Майкла Болтона. Перевіряння — це та частина, яку могла б виконати машина; тестування — це вся діяльність, включно з рішенням, що саме перевіряти, тлумаченням результатів і помічанням того, що ніхто не здогадався перевірити. З незнайомим продуктом більша частина першої години іде на навчання.

RST розрізняє три рівні глибини цієї роботи. **Оглядове тестування (survey testing)** — несистематичне дослідження, мета якого — пізнати продукт і побудувати його мисленнєву модель; воно знаходить дефекти, але стратегії ще не має. **Аналітичне тестування (analysis testing)** — сфокусованіше дослідження, яке виконують, щоб побудувати схему покриття продукту. **Глибоке тестування (deep testing)** — систематична робота за конкретним чартером, зосереджена на ризиках. Починайте з прихильного ставлення: **тур переваг (benefits tour)**, який показує, що продукт робить добре за сприятливих умов, — найшвидший спосіб із ним познайомитися.

Дві ідеї стосуються всього подальшого. Ви ніколи не тестуєте продукт; ви тестуєте свою модель продукту, і погана модель означає погане тестування. І покривати означає робити вибірку: станів у продукту надто багато, щоб їх записати чи виміряти, тому кожен тест — це вибір, куди дивитися, і цей вибір має бути свідомим.

Схема покриття продукту

Схема покриття продукту (product coverage outline, PCO) — це перелік тестопридатних елементів продукту.

За описом Пола Голланда, PCO — це не список тест-кейсів. Вона говорить, *що* можна протестувати, а не *як*. Ви не протестуєте кожен її пункт, і це прийнятно, бо показати, що *не* було протестовано, — само по собі корисна інформація. Схему можна скласти для цілого продукту, для епіка або для однієї функції.

Як її побудувати. Пройдіть продуктом і запишіть, що в ньому є: кожне меню, сторінку, задокументовану кінцеву точку API; кожне поле,

кнопку, фільтр і фрагмент статичного тексту, який вказує на бізнес-правило. Потім пройдіть елементи продукту з Евристичної моделі тестової стратегії (нижче) і запитайте, категорія за категорією, що ви пропустили. Включайте елементи, які не збираєтеся тестувати, щоб ніхто не припустив, що їх покрито. За можливості переглядайте схему разом з іншими; вона змінюється в міру того, як ви дізнаєтеся більше.

Знайдена чи побудована дослідженням. У практичній вправі, описаній Йорісом Меертсом, Гюйбом Схотсом і Рюдом Коксом, дві команди за тридцять хвилин складали схему одного й того самого незнайомого продукту. Одна команда використала довідковий файл продукту, розмічений за категоріями, як *знайдену* схему: повну, але побудовану майже без взаємодії з продуктом. Друга досліджувала продукт і намалювала мапу думок: менше елементів, але краща мисленнєва модель і змістовніші зв'язки між елементами. Жоден із підходів не кращий у загальному випадку. Знайдена схема (документація, опис API, дерево меню) — швидкий старт; дослідження перетворює її на розуміння. Зазначайте, які частини вашої схеми звідки походять. Мапа думок показує зв'язки; список з відступами швидше писати. Працює і те, і те.

Елементи продукту (SFDIPOT)

Евристична модель тестової стратегії (Heuristic Test Strategy Model, HTSM), додаток В до *Taking Testing Seriously*, перелічує сім категорій елементів продукту. Мнемоніка — SFDIPOT.

Категорія	На що дивитися
Структура (Structure)	Код, сервіси та процеси, невиконувані файли (довідка, приклади даних), супутні матеріали (документація, вебсторінки, ліцензії)
Функції (Function)	Багатокористувацькі та паралельні функції; обчислення; поведінка, пов'язана з часом; функції захисту (права, захист даних); перетворення; запуск і зупинка; обробка помилок і кожне повідомлення про помилку; взаємодія функцій; функції тестопридатності (журнали, діагностика)
Дані (Data)	Вхідні та вихідні дані; попередньо задані дані та значення за умовчанням; збережені дані та стан; взаємозалежні дані; послідовності та комбінації; кардинальність (нуль, один, багато, максимум, унікальність); великі та малі обсяги; невалідні дані та шум; життєвий цикл кожної сутності даних (створення, читання, зміна, видалення)

Інтерфейси (Interfaces)	Інтерфейси користувача; системні інтерфейси (журнали, мережа, файли); API; імпорт і експорт
Платформа (Platform)	Зовнішнє апаратне та програмне забезпечення; вбудовані компоненти та бібліотеки; ресурси, які споживає продукт
Операції (Operations)	Види користувачів; середовище; звичайне використання; небажане використання (необізнане, недбале, зловмисне); екстремальне використання
Час (Time)	Час надходження вхідних і появи вихідних даних; швидкий і повільний ввід; зміна темпу (сплески, зависання); паралельність

Критерії якості

Та сама модель перелічує виміри, за якими люди судять про продукт; кожен із них є також категорією ризику. **Здатність (capability)**: чи виконує продукт потрібні функції, достатньо й правильно? **Надійність**: стійкість, обробка помилок, цілісність даних, безпечність. **Зручність**: легкість навчання, легкість роботи, доступність. **Харизма**: естетика, унікальність, образ. **Захищеність (security)**: автентифікація, авторизація, приватність, прогалини в захисті. **Масштабованість**. **Сумісність**: з іншими застосунками, операційними системами, обладнанням, попередніми версіями. **Продуктивність**. **Встановлюваність**. **Розробка (development)**: підтримуваність, тестопридатність, супроводжуваність, переносність, локалізованість.

Рівні покриття

Голланд оцінює покриття якісно, за порядковою шкалою, і обґрунтовує кожен рівень конкретними виконаними тестами. Використовуйте цю шкалу, щоб анотувати схему.

Рівень	Значення
Нічого (nothing)	Не тестувалося
Базова перевірка (sanity)	Один-два основні сценарії; продукт не зламаний повністю
Більше ніж базова (more than sanity)	Кілька простих сценаріїв із помилками, але здебільшого основний сценарій; неглибоко
Типові випадки (common cases)	Різноманіття сценаріїв, складні сценарії та сценарії з помилками; найуживаніші та найкритичніші — глибоко
Деякі межові випадки (some corner cases)	Рідкісні випадки, навантаження, різноманіття даних, великі дані

Глибокі межові випадки (deer corner cases)	Екстремальне різноманіття, тривалі тести, малоімовірні сценарії
--	---

Більшість автоматизованих регресійних перевірок досягає другого рівня і рідко виходить за третій.

Ризик, чартери та сесії

Навмисне тестування керується рішеннями про те, куди найкраще витратити час, і головний чинник тут — **ризик**: наскільки ймовірно, що ця частина продукту відмовить, і наскільки серйозними будуть наслідки? Починайте з обґрунтованих припущень, а потім виконуйте те тестування, яке перетворює припущення на докази. Пройдіть схему й вирішіть, елемент за елементом, чи тестувати його і наскільки глибоко. Якщо представника користувачів немає, поставте себе на їхнє місце; так робить кожен тестувальник продукту версії 1.0.

Чартер (charter) — це коротке формулювання фокуса тестової сесії: місія на наступний блок часу.

Бах запровадив це слово, щоб відрізнити місію сесії від місії всього проєкту. Зручна форма, з вправи Меертса, Схотса та Кокса:

Дослідити ціль за допомогою цих ресурсів або цієї техніки, щоб з'ясувати цю інформацію.

Три приклади для вебзастосунку замовлень та рахунків-фактур:

- Дослідити життєвий цикл замовлення за допомогою дій зі статусом, доступних для одного замовлення, щоб з'ясувати, які переходи дозволені і що відбувається з підсумковими сумами на цьому шляху.
- Дослідити обчислення знижок і доставки за допомогою замовлень, чий проміжні суми близькі до заявлених порогів, щоб з'ясувати, як правила поєднуються і де вони розходяться з описом.
- Дослідити рахунок-фактуру за допомогою замовлень, створених різними видами клієнтів, щоб з'ясувати, чи узгоджується показане в ньому із замовленням, записом клієнта та каталогом.

Сесія походить із сесійного керування тестуванням (session-based test management, SBTM), яке створили Джеймс і Джон Бахи. Сесія — це діяльність, а не документ: практично безперервний блок часу, в якому один або кілька тестувальників працюють над одним чартером. Її елементи:

- **Безперервний час роботи.** Обмежуйте його; зупиняйте відлік на час перерв.
- **Чартер.** Задається до сесії. Якщо сесія відхилилася, бо чартер виявився нездійсненним або з'явилося щось важливіше, перепишіть його після сесії так, щоб він описував, що було зроблено насправді.

- **Звіт.** Чартер, метадані та нотатки. Нотатки — найважливіша частина.
 - **Розподіл часу.** *T-час* (виконання тестів): час, протягом якого ви могли б знайти дефект. *B-час* (дослідження дефектів і звітування про них): час, витрачений на ймовірний дефект; рахується тоді, коли перериває виконання. *S-час* (налаштування та адміністрування): читання, створення даних, конфігурування, нотування; рахується тоді, коли блокує два інші. S-час — нормальна частина сесії; надмірний S-час — сигнал про проблеми тестопридатності.
 - **Розбір.** Дебрифінг після сесії, за можливості — з іншими.
- Аркуш сесії:

Чартер:	Дослідити ... за допомогою ..., щоб з'ясувати ...
Тестувальник:	
Початок / тривалість:	2026-09-20 14:00 / 45 хв
T / B / S:	60 % / 25 % / 15 %
Нотатки:	що покрито, що спостережено, що вирішено і чому
Дефекти:	по одному рядку; звіт пишеться пізніше
Питання:	запитання, перешкоди, те, що потребує когось іншого
Чартер за фактом:	(лише якщо змінився)

Нотування — базова навичка, яка вдосконалюється практикою. Чорнові нотатки швидкі й неохайні і служать кілька годин; архівні нотатки мають дати змогу сторонній людині зараз, або вам через кілька місяців, зрозуміти, яке тестування було виконано. Ведіть чорнові нотатки під час сесії і витратьте п'ятнадцять хвилин наприкінці, щоб перетворити їх на архівні. Добрі нотатки — не перелік натиснутих кнопок. Вони відповідають на питання проєктування тестів: що ви покрили, що виявили, чому це, а не інше, чому ця техніка чи ці дані. Знімки екрана допомагають; знімки всього підряд роблять нотатки нечитабельними.

Дебрифінг перетворює сесії на зв'язну розповідь: кожен тестувальник за кілька хвилин описує сесію і відповідає на запитання; група оцінює, чи виконано чартер, зазначає, що лишилося незрозумілим із нотаток, і записує чартери, які мають бути наступними. Чартер, який потребує більше часу, ніж відведено, — це нормально.

Тури та персони

Тур — простий спосіб думати про вибірку. Приклади RST:

- **Тур інтерфейсом (UI tour).** Знайдіть і відвідайте кожен елемент інтерфейсу користувача; взаємодійте з ним усіма способами, які спадають на думку.
- **Тур клавіатурою (keyboard tour).** Натисніть кожну клавішу та кожну комбінацію клавіш.

- **Тур документацією (documentation tour).** Прочитайте всю документацію користувача і спробуйте кожен приклад.
- **Тур складністю (complexity tour).** Знайдіть частини, де продукт, судячи з усього, виконує багато роботи, і випробуйте їх.
- **Тур даними (data tour).** Знайдіть усі види даних, які можна ввести, імпортувати або з якими можна взаємодіяти, і випробуйте продукт із ними.
- **Тур переваг (benefits tour, співчутливе тестування).** Подивіться, що продукт робить добре за сприятливих умов.
- **Тур помилок (blooper tour).** Користувачі помиляються; пройдіть продуктом, роблячи всі види помилок, які спадають на думку. Наскільки легко втратити дані? Регульовані продукти зобов'язані тестувати на обґрунтовано передбачуване неправильне використання.
- **Тур бізнес-правил (business-rule tour).** Шукайте обмеження та логіку, які запускають певну поведінку за певних умов.

Аналіз оглядового вікна (viewport analysis) належить до турів: розглядайте кожен інтерфейс як обмежений вигляд на певні дані і запитуйте, що ви бачите, що могли б побачити, чи бачите все, що повинні, і нічого з того, чого не повинні. Прокрутіть до кінця, змініть розмір сторінки, подивіться на ті самі дані з правами різних користувачів.

Персона (persona), у розумінні Гаятрі Мохан, — це персонаж, який представляє групу користувачів зі схожими ознаками та потребами. Тестування виконують з погляду користувача, а ви — не користувач: бухгалтер читає рахунок-фактуру інакше, ніж пакувальник на складі; клієнт на повільному мобільному з'єднанні бачить іншу сторінку, ніж менеджер в офісній мережі. Зазвичай продукт покривають від двох до п'яти персон. Оберіть одну на всю сесію і занотуйте, де продукт і персона розходяться.

Чотири шляхи відкриття за Мохан для будь-якої функціональності: **функціональні користувацькі потоки** — спершу позитивний потік одного користувача, потім *повторні потоки* (зробіть те саме двічі; чи помічає це продукт?) і *багатокористувацькі потоки* (двоє користувачів працюють з тими самими даними одночасно); **відмови та обробка помилок** — повільна або відсутня мережа, недоступний сервіс, некоректні дії, і чи пояснює продукт, що пішло не так; **вигляд і поведінка; крос-функціональні аспекти** — захищеність, приватність, автентифікація та авторизація. Дослідження — це не «мавпяче» тестування: воно передбачає точне розуміння того, що функціональність має робити, і звідти досліджує невідоме.

Оракули

Тестовий оракул — це будь-який засіб, за допомогою якого ви розпізнаєте дефект, на який натрапили під час тестування.

Щоразу, коли ви знаходите дефект, запитайте себе, звідки ви знаєте, що це дефект; відповідь і називає оракул. Без оракула у вас є спостереження, підозра або питання, але ще не дефект. Очікувані результати, записані біля кожного кроку, — це один оракул, і вузький; багато оракулів є радше підказками, ніж авторитетними джерелами, і радше неявними, ніж явними. Усі вони ґрунтуються на відношенні узгодженості між продуктом, яким він є, і продуктом, яким він має бути. Дванадцять патернів RST складаються в мнемоніку **A FEW HICCUPPS**:

- **Aspirations (прагнення).** Продукт робить те, що задумано, але зацікавлених осіб можна переконати поставити вищу мету.
- **Familiar problems (знайомі проблеми).** Така поведінка раніше була дефектом, тож, імовірно, є дефектом і тепер.
- **Explainability (пояснюваність).** Поведінку неможливо пояснити. Незрозумілість — привід дослідити.
- **World (світ).** Результат суперечить факту про світ, який представляє програма.
- **History (історія).** Продукт поводиться інакше, ніж раніше, там, де очікувалася сталість.
- **Image (образ).** Продукт не справляє професійного враження; одруки та косметичні вади зменшують довіру.
- **Comparable products (порівнянні продукти).** Функція, порівнянна з функцією іншого продукту, поводиться інакше.
- **Claims (заяви).** Продукт суперечить специфікації, контракту, довідці, рекламі або тому, що сказав розробник.
- **User desires (бажання користувачів).** Поведінка не сподобається користувачам чи іншим, кого вона зачіпає.
- **Product self-consistency (самоузгодженість продукту).** Одна частина продукту суперечить іншій: інтерфейс каже одне, API — інше.
- **Purpose (призначення).** Поведінка не узгоджується з тим, для чого продукт існує.
- **Statutes and standards (закони та стандарти).** Продукт суперечить закону або галузевому стандарту.

Оракули мають кілька **форм**. *Живий оракул* — людина, яка присутня і може сказати «це неправильно». *Явне знання* — те, що така людина сказала, записане. *Неявне знання* — ваше власне уявлення про те, як має поводитися програмне забезпечення, джерело більшості повсякденних знахідок. *Паралельний, або порівнянний, продукт* — оракул через порівняння. *Специфікація* авторитетна, але описує задум, а не кожную

деталь, і живий власник продукту має перевагу над нею. Відчуття — здивування, розгубленість, роздратування — теж оракули: відчуття є сигналом знайти сильніший оракул.

Класи еквівалентності та їхні межі. Підручникове правило каже, що одне значення з кожного класу представляє всі. Бах і Болтон відкидають таке міркування: класи походять з опису того, як продукт *має* працювати, вони розглядають один вимір за раз і не спираються на жодну теорію того, які види дефектів трапляються. Вони називають класи еквівалентності чутками — здогадами про те, що не потребує тестування *саме зараз*. Правило, яке залишається чинним: тестуйте великі відмінності раніше за малі, а потім поглиблюйтеся. Змініть те, що вас цікавить (округлення, форматування, пороги, які зсувають інші правила), — і класи зміняться разом із цим.

Таблиця оракулів:

Спостереження (що ви бачили)	Патерн(и) оракула	Форма
Чому це важливо і для кого	Дефект чи питання?	
Після збереження адреси	Образ,	неявне
Клієнт бачить внутрішню	дефект	
показано «Object reference	пояснюваність	
помилку і не знає, чи		
not set to an instance»		
збереглися дані		
Список товарів відсортовано	Бажання	неявне
Менеджери шукають товар за	питання	
за SKU, а не за назвою	користувачів	
назвою; ніде не сказано, який		
порядок задумано		

Останній стовпчик задає дисципліну: якщо найсильніший оракул, який ви можете назвати, — це неоднозначна заява, запис є питанням до власника продукту, а не звітом про дефект.

Як читати опис продукту як тестувальник

Дефект у вимозі найдешевше знайти до того, як з'явився код. RST називає це **проспективним тестуванням (prospective testing)**: розгляд запропонованої функції або вимоги під час її обговорення з тим самим скептицизмом, який ви застосували б до працюючого продукту. Короткий опис продукту — нагода зробити саме це. Питання зі шпаргалки RST, корисні під час його читання:

1. Чи чітко сформульована вимога? Чи можете ви заплющити очі й точно уявити, як це виглядатиме на практиці?
2. Чи є слова або терміни, які зрозумілі не до кінця або вжиті у двох значеннях?

3. Де опис неоднозначний? Які приклади розв'язали б кожна неоднозначність?
4. Яка головна мета цієї функції, і чи справді описана поведінка її досягне?
5. Чи є доречні галузеві стандарти або нормативні вимоги (виставлення рахунків, податки, персональні дані)?
6. Чи буде функція повністю узгоджена з рештою продукту?
7. Які конкретні дані чи умови вона має обробляти? Які види даних вона зустріне, але не обробить?
8. Які стани або конфігурація мають бути на місці, щоб вона працювала, і що, якщо вони зміняться?
9. Як вона оброблятиме помилки та відновлюватиметься після відмов? Що найгірше може статися: втрата даних, дані, показані не тій людині, непомітне пошкодження даних, фінансові втрати?
10. Як її можна використати неправильно, випадково чи зловмисно? Чи може користувач застрягнути?
11. Як ви отримаєте доступ, отримаєте дані, запустите кожен умовну поведінку і протестуєте функціональність, залежну від часу?
12. Як ви оцінюватимете правильність результату; яким буде оракул?

Заяви — це оракули, а опис — набір заяв. Однозначна заява, якій продукт суперечить, дає дефект. Неоднозначна заява («понад 1000», «перше замовлення», «можна скасувати») дає питання, а фактична поведінка продукту є доказом для обговорення, а не остаточною відповіддю.

Тестопридатність

Практична тестопридатність (testability) — це те, наскільки легко продукт тестувати конкретному тестувальнику з конкретним тестовим процесом у заданому контексті.

Вона є функцією п'яти інших тестопридатностей. **Епістемічна** тестопридатність — розрив між тим, що ви знаєте про якість продукту, і тим, що вам потрібно знати. **Проектна** — фізичні, соціальні та інформаційні умови, в яких ви тестуєте. **Внутрішня (intrinsic)** — властивості самого продукту. **Суб'єктивна** — походить від тестувальника та тестового процесу: навички, знання, стратегія. **Ціннісна** — походить від стандарту якості та того, наскільки добре ви розумієте користувачів.

Слова-підказки (guidewords) з додатка С; внутрішні — повністю:

Вид	Слово-підказка	Значення
Внутрішня	Спостережуваність	Кожен факт про стан і поведінку продукту, а також їхня історія, доступні вам

Внутрішня	Керованість	Ви можете подати будь-який ввід і досягти будь-якого стану чи послідовності станів на вимогу; недетермінованість шкодить
Внутрішня	Алгоритмічна простота	Зв'язок між вводами й виводами можна розглянути та оцінити
Внутрішня	Алгоритмічна прозорість	Хтось знає, як продукт створює свій результат
Внутрішня	Алгоритмічна стабільність	Малі зміни не руйнують логіку докорінно, тож минулі результати лишаються чинними
Внутрішня	Пояснюваність	Дизайн можна пояснити стороннім
Внутрішня	Бездефектність	Дефекти сповільнюють тестування; блокувальні дефекти його зупиняють
Внутрішня	Малість	Менше продукту, менше взаємодій, менше результатів для перегляду
Внутрішня	Розкладність	Частини можна відокремити, тож тестування, дослідження та повторне тестування можна зосередити
Внутрішня	Схожість	Продукт схожий на відому та перевірену технологію
Проектна	Доступність інформації	Ви отримуєте всю інформацію, яка вам потрібна
Проектна	Доступність інструментів	Ви отримуєте потрібні інструменти
Проектна	Пісочниця (sandboxing)	Ви можете виконувати будь-яке варте того тестування, не боячись зашкодити іншим
Проектна	Керованість середовища	Ви контролюєте експериментальні змінні навколо своїх тестів
Ціннісна	Доступність, авторитетність, надійність, точність і дешевизна оракулів	У вас є способи виявити кожен вид проблем, і вони надійні, конкретні та дешеві в застосуванні

Тестувальник мусить вимагати тестопридатності. Ніхто, крім тестувальника, про неї не подумає. Метод Янни Лефлер складається з

трьох кроків. Перший: визначте, що ускладнює тестування — де ви заплуталися, що вас сповільнило, яке слово-підказка це називає: прихований або незадокументований API, неспостережуваний побічний ефект, неможливість скинути дані, відсутність журналу. Другий: з'ясуйте, хто може допомогти це вирішити; за внутрішню тестопридатність відповідають розробники, і вона зводиться здебільшого до керованості, спостережуваності та простоти, а перемикач функції чи діагностична кінцева точка часто є роботою на один день. Третій: поясніть це їхньою мовою: швидший зворотний зв'язок для розробника, простіший код з меншою кількістю дефектів, легше розслідування, коли щось відмовляє. Формулюйте прохання як конкретну дію, а не скаргу: яке спостереження зараз неможливе і яка одна зміна зробила б його можливим.

Інструменти для дослідження вебпродукту

Спершу API. Сторінка OpenAPI (Swagger) — це знайдена схема: кожна кінцева точка, метод, параметр і код відповіді є тестопридатним елементом, а все, чого сторінка *не* перелічує, — питання. Файл `.http`, `curl` або колекція Postman — це запис дослідження: зберігайте кожен запит, який щось показав, з однорядковою нотаткою. API дає змогу зробити те, чого не дозволить інтерфейс: надіслати значення, яке форма відхилила б, викликати операцію, якої немає на сторінці, діяти від імені одного користувача з даними іншого. Саме там продукти й суперечать самі собі.

DevTools браузера, за Мохан: вкладка Console показує помилки сторінки, і сторінка має бути без них; вкладка Network показує кожен запит і відповідь, зокрема те, що інтерфейс надсилає до API, і скільки запитів робить сторінка; «Disable cache» відтворює користувача, який зайшов уперше; обмеження мережі до 3G показує, що робить інтерфейс, поки чекає, і чи шкодить повторне натискання; «Block request URL» імітує недоступний сервіс; вкладка Application показує cookie, які можна змінити або видалити, щоб побачити, як реагує продукт.

Гігієна тестових даних. Застарілі дані заважають дослідженню: замовлення, залишене в незвичному стані попередньою сесією, наступного дня важко пояснити. Починайте кожну сесію з відомих даних — нової бази даних, клієнта, створеного спеціально для цього, зафіксованого початкового стану — і зазначайте цей стан в аркуші сесії. Створення даних — це S-час; якщо воно триває надто довго, це знахідка щодо тестопридатності.