Лекція 2
Тестування програмного забезпечення

2.1. Спрощена класифікація тестування

Тестування можна класифікувати за дуже велику кількість ознак, і практично в кожній серйозній книзі про тестування автор показує свій (який безумовно має право на існування) погляд на це питання.

Відповідний матеріал досить об'ємний і складний, а глибоке розуміння кожного пункту в класифікації вимагає певного досвіду, тому ми розділимо цю тему на дві: зараз ми розглянемо найпростіший, мінімальний набір інформації (рис. 2.1), необхідний тестувальнику-початківцю, а далі наведемо докладну класифікацію .

Рисунок 2.1 - Спрощена класифікація тестування

Отже, тестування можна класифікувати:

1. із запуску коду на виконання;
2. за доступом до коду та архітектури програми;
3. за ступенем автоматизації;
4. за рівнем деталізації застосунку;
5. за ступенем важливості функцій що тестують;
6. за принципами роботи з застосунком;

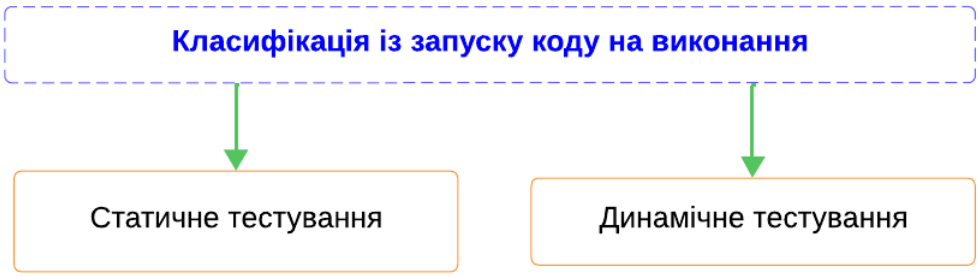
7. за природою застосунку;
8. за фокусуванням на рівні архітектури застосунку;
9. із залучення кінцевих користувачів;
10. за ступенем формалізації;
11. за цілями та завданнями;
12. з технік і підходів;
13. по моменту виконання;

Увага! Дуже часта помилка! Негативні тести не передбачають виникнення у застосунку помилки. Навпаки — вони припускають, що застосунок, що вірно працює, навіть у критичній ситуації поведеться правильним чином (у прикладі з розподілом на нуль, наприклад, відобразить повідомлення «Ділити на нуль заборонено»).

2.2. Детальна класифікація тестування

Наразі ми розглянемо класифікацію тестування максимально докладно. Навіщо взагалі потрібна класифікація тестування? Вона дозволяє впорядкувати знання та значно прискорює процеси планування тестування та розробки тест-кейсів, а також дозволяє оптимізувати трудовитрати за рахунок того, що тестувальнику не доводиться винаходити черговий велосипед.

2.2.1. Класифікація із запуску коду на виконання



Далеко не всяке тестування передбачає взаємодію з працюючим застосунком. Тому в рамках цієї класифікації виділяють:

Статичне тестування – тестування без запуску коду на виконання. В рамках цього підходу тестування можуть піддаватися:

- документи (вимоги, тест-кейси, описи архітектури, схеми баз даних тощо);

- графічні прототипи (наприклад, ескізи інтерфейсу користувача);
- код програми (часто виконується самими програмістами в рамках аудиту коду, що є специфічною варіацією взаємного перегляду до вихідного коду). Код програми також можна перевіряти з використанням технік тестування на основі структур коду;
- параметри налаштування середовища виконання програми;
- підготовлені тестові дані.

Динамічне тестування – тестування із запуском коду на виконання. Запускатися на виконання може:

- код всього застосунку цілком (системне тестування);
- код кількох взаємозалежних частин (інтеграційне тестування);
- окремих частин (модульне чи компонентне тестування);
- окремі ділянки коду.

Основна ідея цього виду тестування полягає в тому, що перевіряється реальна поведінка частин програми.

2.2.2. Класифікація з доступу до коду та архітектури програми

Метод білої скриньки – у тестувальника є доступ до внутрішньої структурі і коду програми, а також є достатньо знань для розуміння побаченого. Виділяють навіть супутню тестуванню методом білої скриньки глобальну техніку — тестування з урахуванням дизайну. Для більш глибокого вивчення суті методу білої скриньки рекомендується ознайомитися з техніками дослідження потоку керування чи потоку даних, використання діаграм станів. Деякі автори схильні жорстко пов'язувати цей метод зі статичним тестуванням, але ніщо не заважає

тестувальнику запустити код на виконання і при цьому періодично звертатися до самого коду (а модульне тестування взагалі передбачає запуск коду на виконання і при цьому роботу саме з кодом, а не з застосунком цілком).

Метод чорної скриньки — у тестувальника або немає доступу до внутрішньої структури і коду програми, або недостатньо знань їхнього розуміння, або він свідомо не звертається до них у процесі тестування. При цьому абсолютна більшість перерахованих на рис. 2.1 видів тестування працюють за методом чорного скриньки, ідею якого в альтернативному визначенні можна сформулювати так: тестувальник робить вплив на застосунок і перевіряє реакцію тим же способом, яким при реальній експлуатації програми на нього впливали б користувачі або інші програми. В рамках тестування за методом чорної скриньки основною інформацією для створення тест-кейсів виступає документація та загальний здоровий глузд (для випадків, коли поведінка застосунку в деякій ситуації не регламентована явно). Іноді це називають «тестуванням на основі неявних вимог», але канонічного визначення у цього підходу немає.

Метод сірої скриньки — комбінація методів білої скриньки та чорної скриньки, яка полягає в тому, що до частини коду та архітектури у тестувальника доступ є, а до частини — ні. На малюнку цей метод позначений особливим пунктиром і сірим кольором тому, що його явна згадка — вкрай рідкісний випадок. Зазвичай говорять про методи білої або чорної скриньки стосовно тих чи інших частин застосунка, при цьому розуміючи, що «застосунок цілком» тестується за методом сірої скриньки.

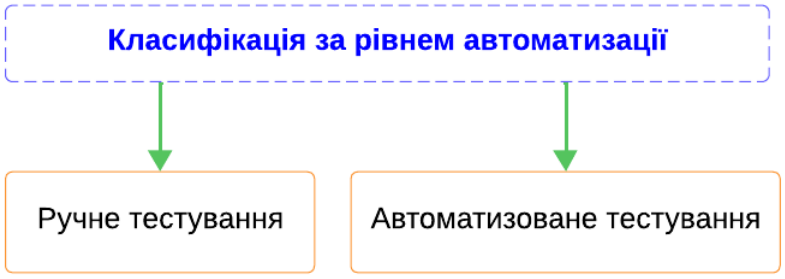
Деякі автори визначають метод сірої скриньки як протиставлення методам білої і чорної скриньки, особливо підкреслюючи, що за методом сірої скриньки внутрішня структура об'єкта, що тестується, відома частково і з'ясовується в міру дослідження. Цей підхід, безперечно, має право на існування, але у своєму граничному разі він вироджується до стану «частину системи ми знаємо, частину — не знаємо», тобто до тієї ж комбінації білої і чорної скриньок.

Методи білої та чорної скриньки не є конкуруючими або взаємовиключними — навпаки, вони гармонійно доповнюють один одного, компенсуючи таким чином недоліки (табл. 2.1).

Таблиця 2.1 - Переваги та недоліки методів білого, чорного та сірого ящиків

	Переваги	Недоліки
Метод білої скриньки	• Показує приховані проблеми та спрощує їх діагностику. • Допускає досить просту автоматизацію тест-кейсів і їх виконання на ранніх стадіях розвитку проекту. • Має розвинену систему метрик, збір та аналіз яких легко автоматизується. • Стимулює розробників до написання якісного коду. • Багато техніки цього методу є перевіреними, добре себе зарекомендували рішеннями, що базуються на строгому технічному підході.	• Не може виконуватися тестувальниками, які не мають достатніх знань в області програмування. • Тестування сфокусовано на реалізованій функціональності, що підвищує ймовірність пропуску нереалізованих вимог. • Поведінка застосунку досліджується у відриві від реального середовища виконання і не враховує її вплив. • Поведінка програми досліджується у відриві від реальних сценаріїв користувачів
Метод чорної скриньки	• Тестувальник не повинен мати (глибокі) знання в галузі програмування. • Поведінка застосунку досліджується в контексті реального середовища виконання та враховує її вплив. • Поведінка програми досліджується в контексті реальних користувальницьких сценаріїв. • Тест-кейси можна створювати на стадії появи стабільних вимог. • Процес створення тест-кейсів дозволяє виявити дефекти в вимогах. • Допускає створення тест-кейсів, які можна багаторазово використовувати на різних проектах.	• Можливе повторення частини тест-кейсів, вже виконаних розробниками. • Висока ймовірність того, що частина можливих варіантів поведінки програми залишиться не протестованою. • Для розробки високоефективних тест-кейсів необхідна якісна документація. • Діагностика виявлених дефектів складніша порівняно з техніками методу білого скриньки. • У зв'язку з широким вибором технік і підходів утруднюється планування і оцінка трудовитрат. • У разі автоматизації можуть знадобитися складні дорогоцінні інструментальні засоби.
Метод сірої скриньки	Поєднує переваги та недоліки методів білої та чорної скриньки.	

2.2.3. Класифікація за рівнем автоматизації



Ручне тестування – тестування, в якому тест-кейси виконуються людиною вручну без використання засобів автоматизації. Незважаючи на те, що це звучить дуже просто, від тестувальника в ті чи інші моменти часу потрібні такі якості, як терплячість, спостережливість, креативність, уміння ставити нестандартні експерименти, а також вміння бачити та розуміти, що відбувається «всередині системи», тобто як зовнішні впливи на застосунок трансформуються у його внутрішні процеси.

Автоматизоване тестування — набір технік, підходів та інструментальних засобів, що дозволяє виключити людину із виконання деяких завдань у процесі тестування. Тест-кейси частково або повністю виконує спеціальний інструментальний засіб, однак розробка тест-кейсів, підготовка даних, оцінка результатів виконання, написання звітів про виявлені дефекти — все це і багато іншого робить людина.

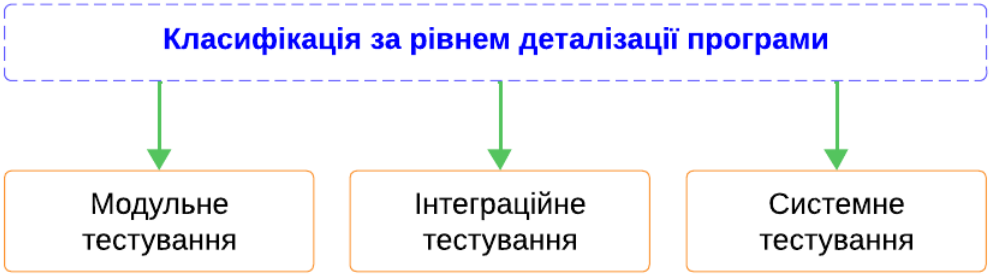
Часто говорять окремо про «напівавтоматизоване» тестування як варіант ручного з частковим використанням засобів автоматизації та окремо про «автоматизоване» тестування (відносячи туди області тестування, в яких комп'ютер виконує відчутно великий відсоток завдань). Але так як без участі людини все одно не обходиться жоден із цих видів тестування, не станемо ускладнювати набір термінів та обмежимося одним поняттям "автоматизоване тестування".

У автоматизованого тестування є багато як сильних, і слабких сторін (табл. 2.2).

Таблиця 2.2 - Переваги та недоліки автоматизованого тестування

Переваги	Недоліки
Швидкість виконання тест-кейс може в рази і на порядки перевищувати можливості людини.	Необхідний висококваліфікований персонал через те, що автоматизація — це «проект усередині проекту» (зі своїми вимогами, планами, кодом тощо.)
Відсутність впливу людського фактору у процесі виконання тест-кейсів (втоми, неуважності тощо).	Високі витрати на складні засоби автоматизації, розробку та супровід коду тест-кейсів.
Мінімізація витрат при багаторазовому виконанні тест-кейсів (участь людини тут потрібна лише епізодично).	Автоматизація потребує більш ретельного планування та управління ризиками, так як в іншому випадку проекту може бути завдано серйозної шкоди.
Здатність засобів автоматизації виконати тест-кейси, в принципі непосильні для людини через свою складність, швидкість або інші фактори.	Коштує автоматизації дуже багато, що може спричинити фінансові витрати (і ризики), необхідність навчання персоналу (або пошуку фахівців).
Здатність засобів автоматизації збирати, зберігати, аналізувати, агрегувати і представляти в зручній для сприйняття людиною формі колосальні обсяги даних.	У разі відчутної зміни вимог, зміни технологічного домену, переробки інтерфейсів (як користувацьких, так і програмних) багато тест-кейсів стають безнадійно застарілими і вимагають створення заново.
Здатність засобів автоматизації виконувати низькорівневі дії із застосунком, операційною системою, каналами передачі і т.д.	Якщо ж висловити всі переваги та недоліки автоматизації тестування однією фразою, то виходить, що автоматизація дозволяє відчутно збільшити тестове покриття, але при цьому відчутно збільшує ризики.

2.2.4. Класифікація за рівнем деталізації програми



Модульне тестування спрямоване на перевірку окремих невеликих частин програми, які зазвичай можна досліджувати ізольовано від інших подібних частин. За виконання цього тестування можуть перевірятися окремі функції чи методи класів, самі класи, взаємодія класів, невеликі бібліотеки, окремі частини застосунка. Часто

даний вид тестування реалізується з використанням спеціальних технологій та інструментальних засобів автоматизації тестування, що значно спрощують та прискорюють розробку відповідних тест-кейсів.

Через особливості перекладу українською мовою втрачаються нюанси ступеня деталізації: «юніт-тестування», як правило, спрямоване на тестування атомарних ділянок коду, «модульне» — на тестування класів та невеликих бібліотек, «компонентне» — на тестування бібліотек та структурних частин програми. Але ця класифікація не стандартизована, і в різних авторів можна зустріти різні взаємовиключні трактування.

Інтеграційне тестування спрямоване на перевірку взаємодії між декількома частинами програми (кожна з яких, у свою чергу, перевірена окремо на стадії модульного тестування). На жаль, навіть якщо ми працюємо з дуже якісними окремими компонентами, «на стику» їхньої взаємодії часто виникають проблеми. Саме ці проблеми виявляє інтеграційне тестування.

Системне тестування спрямовано на перевірку всього застосунка як єдиного цілого, зібраного з частин, перевірених на двох попередніх стадіях. Тут якраз виявляються дефекти «на стиках» компонентів, з'являється можливість повноцінно взаємодіяти з застосунком з погляду кінцевого користувача, застосовуючи безліч інших видів тестування.

З класифікацією за рівнем деталізації застосунка пов'язаний цікавий сумний факт: якщо попередня стадія виявила проблеми, то на наступній стадії ці проблеми точно завдадуть удару по якості; якщо ж попередня стадія не виявила проблем, це ще аж ніяк не захищає нас від проблем на наступній стадії.

2.2.5. Класифікація за ступенем важливості функцій, що тестуються

У деяких джерелах цей різновид класифікації також називають «по глибині тестування».

Димове тестування спрямоване на перевірку найважливішої функціональності, непрацездатність якої робить безглуздою саму ідею використання програми (або іншого об'єкта, що піддається димовому тестуванню).

Димове тестування проводиться після виходу нового білда, щоб визначити загальний рівень якості програми та ухвалити рішення про доцільність виконання тестування критичного шляху та розширеного тестування. Оскільки тест-кейсів на рівні димового тестування відносно небагато, а самі вони досить прості, але дуже часто повторюються, вони є хорошими кандидатами на автоматизацію. У зв'язку з високою важливістю тест-кейсів на цьому рівні граничне значення метрики їх проходження часто виставляється рівним 100% або близьким до 100%.

Тестування критичного шляху спрямовано на дослідження функціональності, використовуваної типовими користувачами у типової повсякденної діяльності. Як видно з визначення англомовної версії терміна, сама ідея запозичена з управління проектами та трансформована в контексті тестування на наступну: існує більшість користувачів, які найчастіше використовують певну підмножину функцій програми. Саме ці функції і потрібно перевірити, щоб переконатися, що застосунок «в принципі працює». Якщо з якихось причин програма не виконує ці функції або виконує їх некоректно, багато користувачів не зможуть досягти безлічі своїх цілей. Порогове значення метрики успішного проходження «тесту критичного шляху» вже трохи нижче, ніж у димовому тестуванні, але все одно досить високе (як правило, близько 70 – 90 % — залежно від суті проекту).

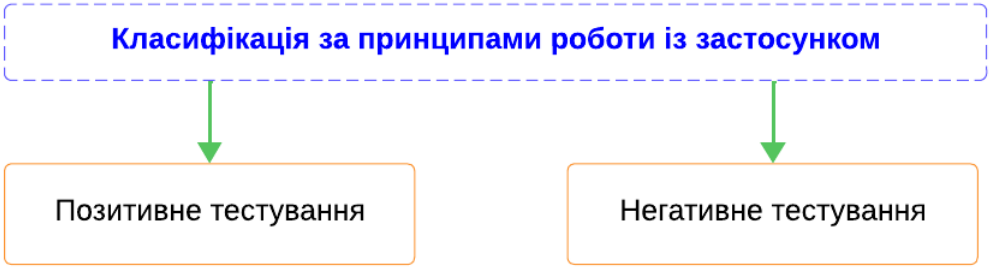
Розширене тестування спрямоване на дослідження всієї заявленої у вимогах функціональності — навіть тієї, яка низько проранжована за рівнем ваги. При цьому тут також враховується, яка функціональність є більш важливою, а яка менш важливою. Але за наявності достатньої кількості часу та інших ресурсів тест-кейси цього рівня можуть торкнутися навіть найнижчих пріоритетних вимог.

Ще одним напрямом дослідження в рамках даного тестування є нетипові, малоймовірні, екзотичні випадки та сценарії використання функцій та властивостей застосунку, які торкнулися попередніх рівнів. Порогове значення метрики успішного проходження розширеного

тестування істотно нижче, ніж у тестуванні критичного шляху (іноді можна побачити навіть значення в діапазоні 30 – 50 %, тому що переважна більшість знайдених дефектів не становить загрози для успішного використання програми більшістю користувачів).

На жаль, часто можна зустріти думку, що димове тестування, тестування критичного шляху і розширене тестування безпосередньо пов'язані з позитивним тестуванням і негативним тестуванням, і негативне з'являється лише на рівні тестування критичного шляху. Це не так. Як позитивні, і негативні тести можуть (і інколи й мають) зустрічатися на всіх перелічених рівнях. Наприклад, розподіл на нуль у калькуляторі явно має відноситися до димового тестування, хоча це яскравий приклад негативного тест-кейсу.

2.2.6. Класифікація за принципами роботи із застосунком

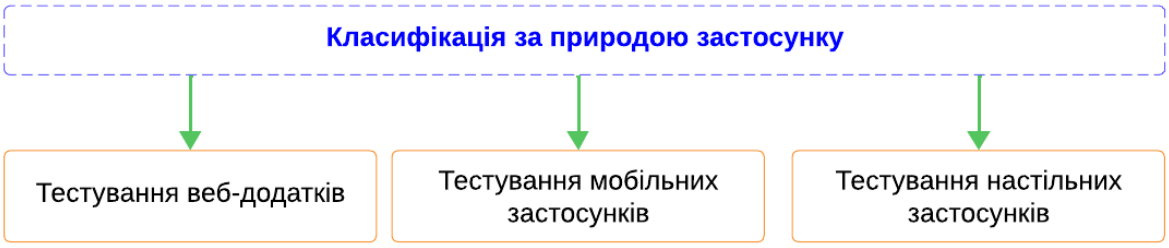


Позитивне тестування спрямоване на дослідження програми в ситуації, коли всі дії виконуються суворо за інструкцією без будь-яких помилок, відхилень, введення невірних даних і т. п. Якщо позитивні тест-кейси завершуються помилками, це тривожна ознака — програма працює невірно навіть в ідеальних умовах (і можна припустити, що в неідеальних умовах вона працює ще гірше). Для прискорення тестування кілька позитивних тест-кейсів можна поєднувати (наприклад, перед відправкою заповнити всі поля форми правильними значеннями) — іноді це може ускладнити діагностику помилки, але суттєва економія часу компенсує цей ризик.

Негативне тестування спрямоване на дослідження роботи програми у ситуаціях, коли з ним виконуються (некоректні) операції або використовуються дані, які потенційно призводять до помилок (класика жанру — розподіл на нуль). Оскільки в реальному житті таких ситуацій значно більше (користувачі припускаються помилок, зловмисники

усвідомлено «ламають» застосунок, у середовищі роботи програми виникають проблеми і т. п.), негативних тест-кейсів виявляється значно більше, ніж позитивних (іноді — в рази чи навіть на порядки). На відміну від позитивних, негативні тест-кейси не варто об'єднувати, так як подібне рішення може призвести до неправильного трактування поведінки програми та пропуску (не виявлення) дефектів.

2.2.7. Класифікація за природою застосунку



Даний вид класифікації є штучним, оскільки «всередині» йтиметься про ті самі види тестування, що відрізняються в даному контексті лише концентрацією на відповідних функціях та особливостях застосунка, використанням специфічних інструментів та окремих технік.

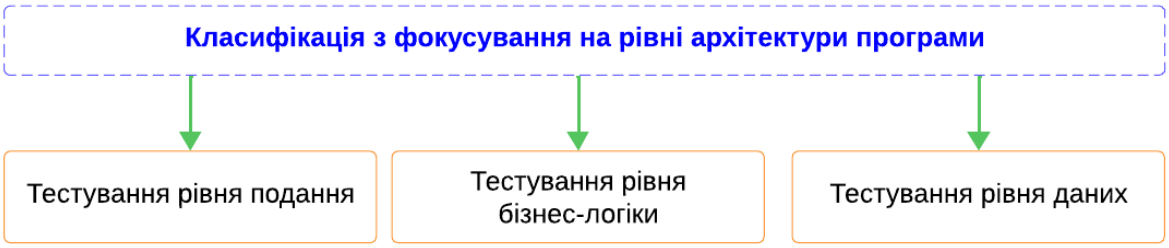
Тестування веб-додатків пов'язане з інтенсивною діяльністю в галузі тестування сумісності (особливо крос-браузерного тестування), тестування продуктивності, автоматизації тестування з використанням широкого спектру інструментальних засобів.

Тестування мобільних застосунків також потребує підвищеної уваги до тестування сумісності, оптимізації продуктивності (у тому числі клієнтської частини з точки зору зниження енергоспоживання), автоматизації тестування із застосуванням емуляторів мобільних пристроїв.

Тестування настільних додатків є найкласичнішим серед усіх перелічених у даній класифікації, та його особливості залежать від предметної галузі застосунка, нюансів архітектури, ключових показників якості тощо.

Цю класифікацію можна продовжувати дуже довго. Наприклад, можна окремо розглядати тестування консольних додатків та додатків з графічним інтерфейсом, серверних додатків та клієнтських додатків тощо.

2.2.8. Класифікація з фокусування на рівні архітектури програми



Цей вид класифікації, як і попередній, також є штучним і відображає лише концентрацію уваги на окремій частині програми.

Тестування рівня подання сконцентровано на тій частині програми, яка відповідає за взаємодію із «зовнішнім світом» (як користувачами, так і іншими програмами). Досліджуються питання зручності використання, швидкості відгуку інтерфейсу, сумісності з браузерами, коректності роботи інтерфейсів.

Тестування рівня бізнес-логіки відповідає за перевірку основного набору функцій програми та будується на базі ключових вимог до застосунку, бізнес-правил та загальної перевірки функціональності.

Тестування рівня даних сконцентровано на тій частині програми, яка відповідає за зберігання та деяку обробку даних (найчастіше — у базі даних чи іншому сховищі). Тут особливий інтерес є тестування даних, перевірка дотримання бізнес-правил, тестування продуктивності.

2.2.9. Класифікація із залучення кінцевих користувачів

Всі три перелічені нижче види тестування відносяться до операційного тестування .

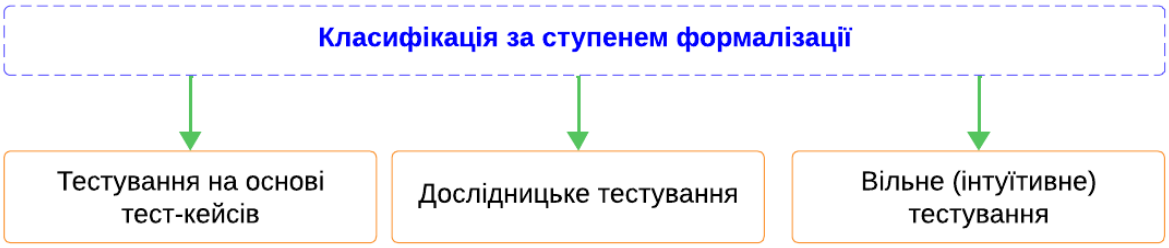
Альфа-тестування виконується всередині організації-

розробника з можливим частковим залученням кінцевих користувачів. Може бути формою внутрішнього приймального тестування. У деяких джерелах наголошується, що це тестування має проводитися без залучення команди розробників, але інші джерела не висувають такої вимоги. Суть цього виду коротко: продукт вже можна періодично показувати зовнішнім користувачам, але він ще досить «сирий», тому основне тестування виконується організацією-розробником.

Бета-тестування виконується організацією-розробником з активним залученням кінцевих користувачів. Може бути формою зовнішнього приймального тестування. Суть цього виду коротко: продукт вже можна відкрито показувати зовнішнім користувачам, він вже досить стабільний, але проблеми все ще можуть бути, і для їх виявлення потрібний зворотний зв'язок від реальних користувачів.

Гамма-тестування - фінальна стадія тестування перед випуском продукту, спрямована на виправлення незначних дефектів, виявлених у бета-тестуванні. Як правило, також виконується з максимальним залученням кінцевих користувачів/замовників. Може бути формою зовнішнього приймального тестування. Суть цього виду коротко: продукт вже майже готовий, і зараз зворотний зв'язок від реальних користувачів використовується для усунення останніх недоробок.

2.2.10. Класифікація за ступенем формалізації



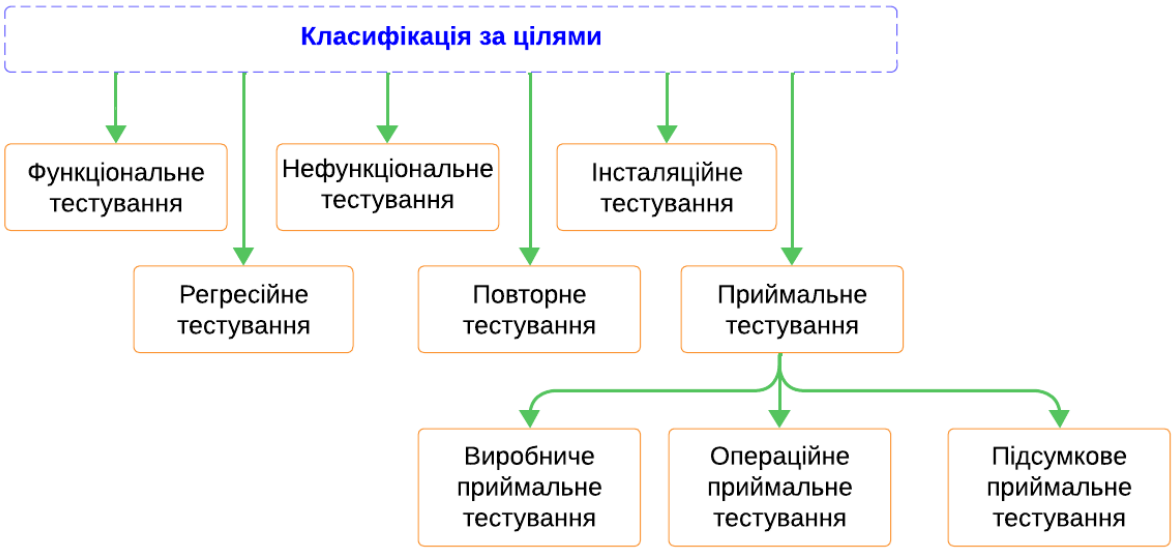
Тестування на основі тест-кейсів - формалізований підхід, в якому тестування проводиться на основі заздалегідь підготовлених тест-кейсів, наборів тест-кейсів та іншої документації. Це найпоширеніший спосіб тестування, який також дозволяє досягти максимальної повноти дослідження програми за рахунок суворої систематизації процесу, зручності застосування метрик та широкого набору вироблених за десятиліття та перевірених на практиці

рекомендацій.

Дослідницьке тестування - частково формалізований підхід, в рамках якого тестувальник виконує роботу з застосунком за вибраним сценарієм, який, у свою чергу, допрацьовується в процесі виконання з метою повнішого дослідження програми. Ключовим фактором успіху при виконанні дослідницького тестування є робота за сценарієм, а не виконання розрізнених бездумних операцій. Існує навіть спеціальний сценарний підхід, який називають сесійним тестуванням. В якості альтернативи сценаріям при виборі дій з застосунком іноді можуть використовуватися чек-листи, і тоді цей вид тестування називають тестуванням на основі чек-листів.

Вільне (інтуїтивне) тестування — повністю неформалізований підхід, у якому не передбачається використання ні тест-кейсів, ні чек-листів, ні сценаріїв — тестувальник повністю спирається на свій професіоналізм та інтуїцію для спонтанного виконання з застосунком дій, які, як він вважає, можуть виявити помилку. Цей вид тестування використовується рідко і виключно як доповнення до повністю або частково формалізованого тестування у випадках, коли для дослідження певного аспекту поведінки програми немає тест-кейсів.

2.2.11. Класифікація за цілями



Функціональне тестування — вид тестування, спрямований на перевірку коректності роботи функціональності застосунку. Часто функціональне тестування асоціюють з тестуванням за методом чорної

скриньки, однак і за методом білої скриньки можна перевіряти коректність реалізації функціональності.

Часто виникає питання, у чому різниця між функціональним тестуванням та тестуванням функціональності. Якщо коротко, то:

функціональне тестування спрямоване на перевірку того, які функції програми реалізовані, і що вони працюють правильно;

тестування функціональності спрямоване на теж завдання, але акцент зміщений у бік дослідження застосунка в реальному робочому середовищі.

Нефункціональне тестування - вид тестування, спрямований на перевірку нефункціональних особливостей застосунку, таких як зручність використання, сумісність, продуктивність, безпека тощо.

Інсталяційне тестування — тестування, спрямоване на виявлення дефектів, які виникають на стадії інсталяції застосунку. Загалом таке тестування перевіряє безліч сценаріїв та аспектів роботи інсталятора в таких ситуаціях, як:

нове середовище виконання, в якій застосунок раніше не був інстальований;

оновлення існуючої версії («апгрейд»);

зміна поточної версії на більш стару («даунгрейд»);

повторна установка програми з метою усунення виниклих проблем («переінсталяція»);

повторний запуск інсталяції після помилки, що призвела до неможливості продовження інсталяції;

видалення програми;

встановлення нової програми із сімейства додатків;

автоматична інсталяція без участі користувача.

Регресійне тестування - тестування, спрямоване на перевірку того факту, що в раніше працездатній функціональності не з'явилися помилки, спричинені змінами у застосунку або середовищі його функціонування. Фундаментальна проблема у супроводі програм полягає в тому, що виправлення однієї помилки з великою ймовірністю (20 – 50 %) спричиняє появу нової. Тому регресійне тестування є невід'ємним інструментом забезпечення якості та активно використовується практично у будь-якому проекті.

Повторне тестування — виконання тест-кейсів, які раніше виявили дефекти з метою підтвердження усунення дефектів. Фактично

цей вид тестування зводиться до дій на фінальній стадії життєвого циклу звіту про дефект, спрямованим на те, щоб перевести дефект у стан перевірений і закритий.

Приймальне тестування - формалізоване тестування, спрямоване на перевірку програми з точки зору кінцевого користувача/замовника та винесення рішення про те, чи приймає замовник роботу у виконавця (проектної команди). Можна виділити такі підвиди приймального тестування (хоча згадують їх вкрай рідко, обмежуючись в основному загальним терміном «приймальне тестування»):

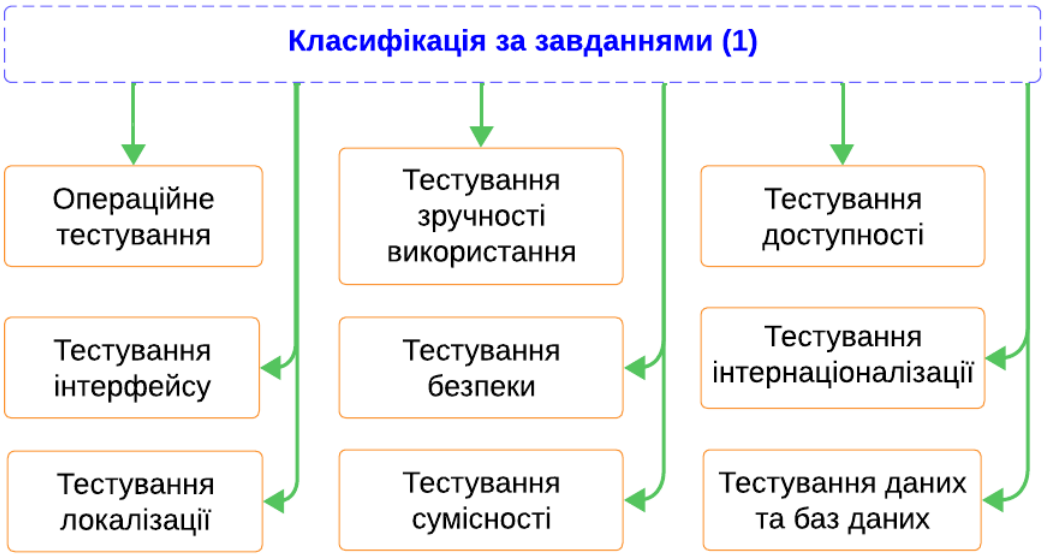
- *Виробниче приймальне тестування* — дослідження повноти та якості реалізації програми, що виконується проектною командою, з точки зору його готовності до передачі замовнику. Цей вид тестування часто розглядається як синонім альфа-тестування.
- *Операційне приймальне тестування* - операційне тестування, яке виконується з точки зору виконання інсталяції, споживання застосунком ресурсів, сумісності з програмною та апаратною платформою і т. п.
- Підсумкове приймальне тестування - тестування кінцевими користувачами (представниками замовника) програми в реальних умовах експлуатації з метою винесення рішення про тому, чи вимагає застосунок або може бути прийнято в експлуатацію в поточному вигляді.

2.2.12. Класифікація за завданнями (1)

Операційне тестування — тестування, яке проводиться в реальному або наближеному до реального операційному середовищі, що включає операційну систему, системи управління базами даних, сервери додатків, веб-сервери, апаратне забезпечення і т. п.

Тестування зручності використання - тестування, спрямоване на дослідження того, наскільки кінцевому користувачеві зрозуміло, як працювати з продуктом, а також те, наскільки йому подобається використовувати продукт. І це не застереження - дуже часто успіх продукту залежить саме від емоцій, які він викликає у користувачів. Для ефективного проведення цього виду тестування потрібно реалізувати

досить серйозні дослідження із залученням кінцевих користувачів, проведенням маркетингових досліджень тощо.



Тестування зручності використання (usability testing) та тестування інтерфейсу користувача (GUI testing) – не одне й те саме! Наприклад, інтерфейс може бути незручним, а зручний може працювати некоректно.

Тестування доступності - тестування, спрямоване на дослідження придатності продукту до використання людьми з обмеженими можливостями (слабким зором і т. п.).

Тестування інтерфейсу — тестування, спрямоване на перевірку інтерфейсів програми або її компонентів. За визначенням ISTQB-глосарія цей вид тестування відноситься до інтеграційного тестування, і це цілком справедливо для таких його варіацій як тестування інтерфейсу прикладного програмування та інтерфейсу командного рядка, хоча останнє може виступати і як різновид тестування інтерфейсу користувача, якщо через командний рядок з застосунком взаємодіє користувач, а не інший застосунок.

Тестування безпеки — тестування, спрямоване на перевірку здатності програми протистояти зловмисним спробам отримання доступу до даних або функцій, права на доступ до яких зловмисник не має.

Тестування інтернаціоналізації - тестування, спрямоване на перевірку готовності продукту до роботи з використанням різних мов та з урахуванням різних національних та культурних особливостей. Цей

вид тестування не має на увазі перевірки якості відповідної адаптації (цим займається тестування локалізації), воно сфокусоване саме на перевірці можливості такої адаптації (наприклад: що буде, якщо відкрити файл з ієрогліфом в імені; як буде працювати інтерфейс, якщо все перекласти на японський; чи може застосунок шукати дані в тексті корейською).

Тестування локалізації — тестування, спрямоване на перевірку коректності та якості адаптації продукту до використання тією чи іншою мовою з урахуванням національних та культурних особливостей. Це тестування слідує за тестуванням інтернаціоналізації та перевіряє коректність перекладу та адаптації товару, а не готовність товару до таких процесів.

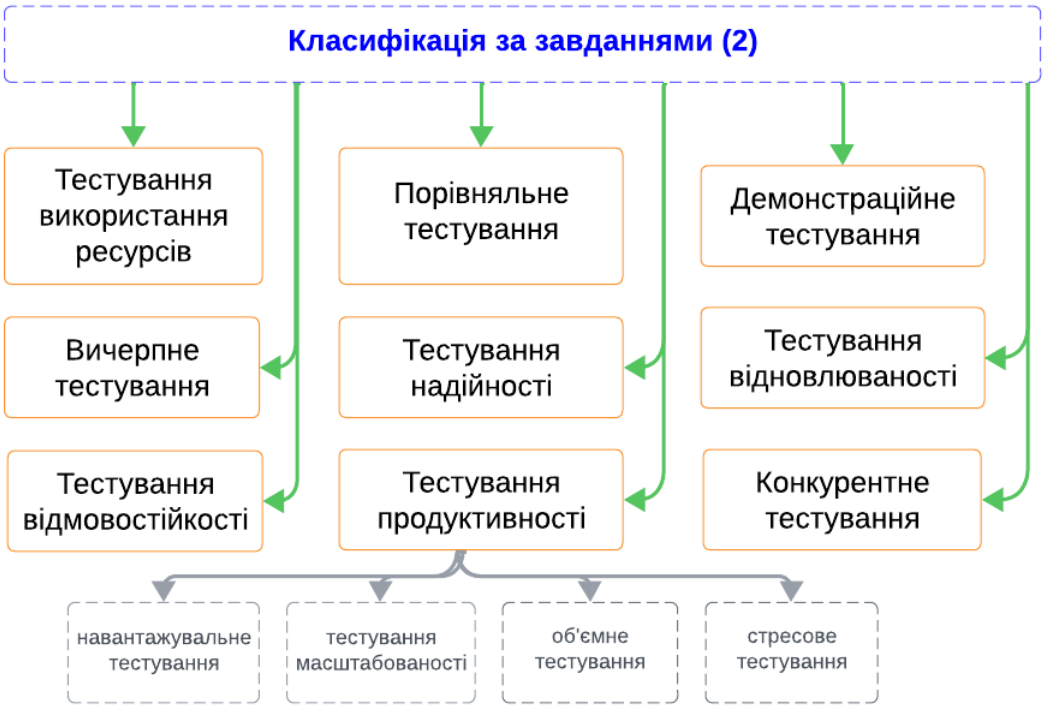
Тестування сумісності - тестування, спрямоване на перевірку здатності програми працювати в зазначеному оточенні. Тут, наприклад, може перевірятись:

- сумісність з апаратною платформою, операційною системою та мережевою інфраструктурою (конфігураційне тестування);
- сумісність з браузерами та їх версіями (крос-браузерне тестування);
- сумісність із мобільними пристроями;
- і так далі.

Тестування даних та баз даних — два близькі за змістом види тестування, спрямовані на дослідження таких характеристик даних, як повнота, несуперечність, цілісність, структурованість тощо. У контексті баз даних дослідженню може піддаватися адекватність моделі предметної області, здатність моделі забезпечувати цілісність і консистентність даних, коректність роботи тригерів, процедур, що зберігаються і т. п.

2.2.13. Класифікація за завданнями (2)

Тестування використання ресурсів – сукупність видів тестування, що перевіряють ефективність використання застосунком доступних йому ресурсів та залежність результатів роботи програми від кількості доступних йому ресурсів. Часто ці види тестування прямо чи опосередковано примикають до техніки тестування продуктивності.



Порівняльне тестування – тестування, спрямоване на порівняльний аналіз переваг і недоліків продукту, що розробляється по відношенню до його основних конкурентів.

Демонстраційне тестування – формальний процес демонстрації замовнику продукту з підтвердження, що відповідає усім заявленим вимогам. На відміну від приймального тестування цей процес суворіший і всеосяжний, але може проводитися і на проміжних стадіях розробки продукту.

Вичерпне тестування – тестування програми з усіма можливими комбінаціями всіх можливих вхідних даних у всіх можливих умовах виконання. Для складної системи не може бути реалізовано, але може застосовуватися для перевірки окремих украй простих компонентів.

Тестування надійності – тестування здатності програми виконувати свої функції в заданих умовах протягом заданого часу або заданої кількості операцій.

Тестування відновлюваності – тестування здатності програми відновлювати свої функції та заданий рівень продуктивності, а також відновлювати дані у разі виникнення критичної ситуації, що призводить до тимчасової (часткової) втрати працездатності програми.

Тестування відмовостійкості – тестування, що полягає в емуляції або реальному створенні критичних ситуацій з метою

перевірки здатності застосунку задіяти відповідні механізми, що запобігають порушення працездатності, продуктивності та пошкодження даних.

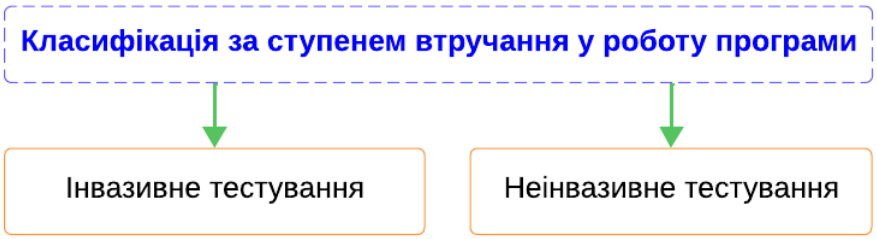
Тестування продуктивності – дослідження показників швидкості реакції застосунку на зовнішні впливи за різного характеру та інтенсивності навантаження. В рамках тестування продуктивності виділяють такі підвиди:

- *навантажувальне тестування* – дослідження здатності програми зберігати задані показники якості при навантаженні в допустимих межах і деякому перевищенні цих меж (визначення "запасу міцності").
- *тестування масштабованості* - дослідження здатності програми збільшувати показники продуктивності відповідно до збільшення кількості доступних застосунку ресурсів.
- *об'ємне тестування* - дослідження продуктивності програми при обробці різних (як правило, великих) обсягів даних.
- *стресове тестування* - дослідження поведінки програми при позаштатних змінах навантаження, що значно перевищують розрахунковий рівень, або в ситуаціях недоступності значної частини необхідних застосунку ресурсів. Стресове тестування може виконуватися і поза контекстом навантажувального тестування: тоді воно, як правило, називається «тестуванням на руйнування» і являє собою крайню форму негативного тестування.

Конкурентне тестування — дослідження поведінки програми у ситуації, коли йому доводиться обробляти велику кількість запитів, що одночасно надходять, що викликає конкуренцію між запитами за ресурси (базу даних, пам'ять, канал передачі даних, дискову підсистему тощо). Іноді під конкурентним тестуванням розуміють дослідження роботи багатопоточних додатків і коректність синхронізації дій, вироблених у різних потоках.

Як окремі або допоміжні техніки в рамках тестування продуктивності можуть використовуватися тестування використання ресурсів, тестування надійності, тестування відновлюваності, тестування відмовостійкості і т.д.

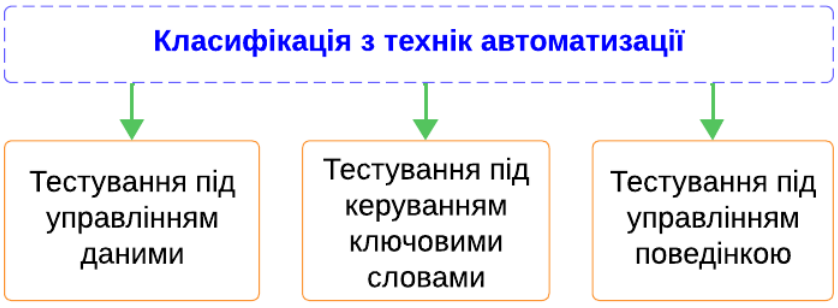
2.2.14. Класифікація за ступенем втручання у роботу програми



Інвазивне тестування – тестування, виконання якого може вплинути на функціонування програми через роботу інструментів тестування (наприклад, будуть спотворені показники продуктивності) або через втручання в сам код програми (наприклад, для аналізу роботи програми було додано додаткове протоколювання, включений висновок налагоджувальної інформації та і т. п.). Деякі джерела розглядають інвазивне тестування як форму негативного чи навіть стресового тестування.

Неінвазивне тестування – тестування, виконання якого непомітно для застосування і не впливає на процес його нормальної роботи.

2.2.15. Класифікація з технік автоматизації



Тестування під управлінням даними - спосіб розробки автоматизованих тест-кейсів, в якому вхідні дані та очікувані результати виносяться за межі тест-кейсу і зберігаються поза ним - у файлі, базі даних і т. п.

Тестування під керуванням ключовими словами - спосіб розробки автоматизованих тест-кейсів, в якому за межі тест-кейсу виноситься не тільки набір вхідних даних та очікуваних результатів, а й логіка

поведінки тест-кейс, яка описується ключовими словами (командами).

Тестування під управлінням поведінкою — спосіб розробки автоматизованих тест-кейсів, у якому основна увага приділяється коректності роботи бізнес-сценаріїв, а не окремим деталям функціонування застосунку.

2.2.16. Класифікація на основі джерел помилок

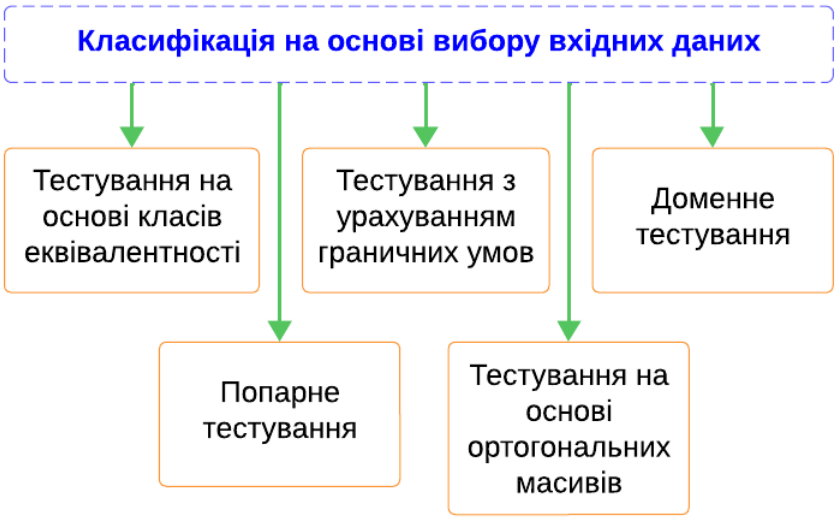
Тестування передбаченням помилок – техніка тестування, при якої тести розробляються на основі досвіду тестувальника та його знань про те, які дефекти типові для тих чи інших компонентів чи областей функціональності програми. Може комбінуватися з технікою т.зв. «помилка орієнтованого» тестування, в якому нові тести будуються на основі інформації про раніше виявлені в застосунку проблеми.

Евристична оцінка – техніка тестування зручності використання, спрямована на пошук проблем в інтерфейсі користувача, що є відхиленням від загальноприйнятих норм.

Мутаційне тестування – техніка тестування, в якій порівнюється поведінка кількох версій того самого компонента, причому частина таких версій може бути спеціально розроблена з додаванням помилок (що дозволяє оцінити ефективність тест-кейсів — якісні тести виявлять ці спеціально додані помилки). Може комбінуватися з наступним у списку видом тестування (тестуванням додаванням помилок).

Тестування додаванням помилок – техніка тестування, при якій в застосунок спеціально додаються заздалегідь відомі, спеціально продумані помилки з метою моніторингу їх виявлення та усунення та, таким чином, формування більш точної оцінки показників процесу тестування.

2.2.17. Класифікація на основі вибору вхідних даних



Тестування на основі класів еквівалентності – техніка тестування, спрямована на скорочення кількості тест-кейсів, що розробляються і виконуються при збереженні достатнього тестового покриття. Суть техніки полягає у виявленні наборів еквівалентних тест-кейсів (кожен з яких перевіряє одну й ту саму поведінку застосунка) та виборі з таких наборів невеликої підмножини тест-кейсів, що з найбільшою ймовірністю виявлять проблему.

Тестування з урахуванням граничних умов — інструментальна техніка тестування з урахуванням класів еквівалентності, що дозволяє виявити специфічні значення досліджуваних параметрів, які стосуються меж класів еквівалентності. Ця техніка значно полегшує виявлення наборів еквівалентних тест-кейсів та вибір таких тест-кейсів, які виявлять проблему з найбільшою ймовірністю.

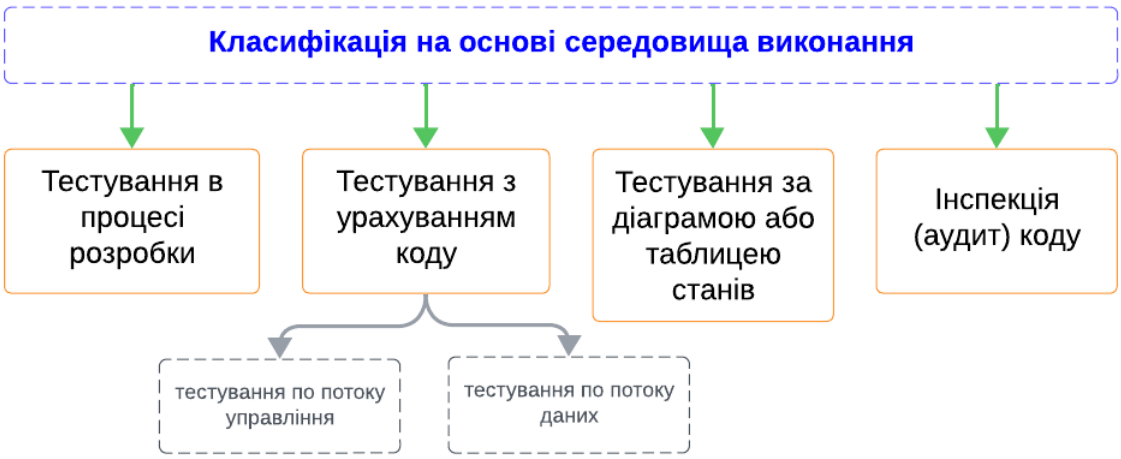
Доменне тестування – техніка тестування на основі класів еквівалентності та граничних умов, що дозволяє ефективно створювати тест-кейси, що зачіпають кілька параметрів (змінних) одночасно (у тому числі з урахуванням взаємозалежності цих параметрів). Дана техніка також описує підходи до вибору мінімальної множини показових тест-кейсів з усього набору можливих тест-кейсів.

Попарне тестування – техніка тестування, в якій тест-кейси будуються за принципом перевірки пар значень параметрів (змінних), замість того, щоб намагатися перевірити всі можливі комбінації всіх значень усіх параметрів. Ця техніка є окремим випадком N-

комбінаційного тестування і дозволяє істотно скоротити трудовитрати на тестування (а іноді і взагалі уможливити тестування у випадку, коли кількість «всіх комбінацій усіх значень усіх параметрів» вимірюється мільярдами).

Тестування на основі ортогональних масивів – інструментальна техніка попарного і N-комбінаційного тестування, що ґрунтується на використанні т.зв. «ортогональних масивів» (двовимірних масивів, що мають наступну властивість: якщо взяти дві будь-які колонки такого масиву, то «підмасив», що вийшов, міститиме всі можливі попарні комбінації значень, представлених у вихідному масиві).

2.2.18. Класифікація на основі середовища виконання



Тестування в процесі розробки – тестування, яке виконується безпосередньо в процесі розробки програми або в середовищі виконання, відмінного від середовища реального використання програми. Зазвичай, виконується самими розробниками.

Тестування з урахуванням коду – у різних джерелах цю техніку називають по-різному (найчастіше тестуванням на основі структур, причому деякі автори змішують в один набір тестування по потоку управління і по потоку даних, а деякі суворо поділяють ці стратегії). Підвиди цієї техніки також організують у різні комбінації, але найбільш універсально їх можна класифікувати так:

- *тестування по потоку управління* – сімейство технік тестування, в яких тест-кейси розробляються з метою активації та перевірки виконання різних послідовностей подій, що

визначаються за допомогою аналізу вихідного коду програми.

- *тестування по потоку даних* – сімейство технік тестування, заснованих на виборі окремих шляхів з потоку управління з метою дослідження подій, пов'язаних зі зміною стану змінних.

Тестування за діаграмою або таблицею станів – техніка тестування, в якій тест-кейси розробляються для перевірки переходів програми з одного стану до іншого. Стани можуть бути описані діаграмою або таблицею.

Іноді цю техніку тестування також називають тестуванням за принципом кінцевого автомата. Важливою перевагою цієї техніки є можливість застосування в ній теорії кінцевих автоматів (які добре формалізовані), а також можливість використання автоматизації для створення комбінацій вхідних даних.

Інспекція (аудит) коду – сімейство технік підвищення якості коду за рахунок того, що в процесі створення або вдосконалення коду беруть участь кілька людей. Ступінь формалізації аудиту коду може змінюватись від досить швидкого перегляду до ретельної формальної інспекції. На відміну від технік статичного аналізу коду (по потоку управління та потоку даних) аудит коду також покращує такі його характеристики, як зрозумілість, підтримуваність, відповідність угод про оформлення і т. п. Аудит коду виконується переважно самими програмістами.

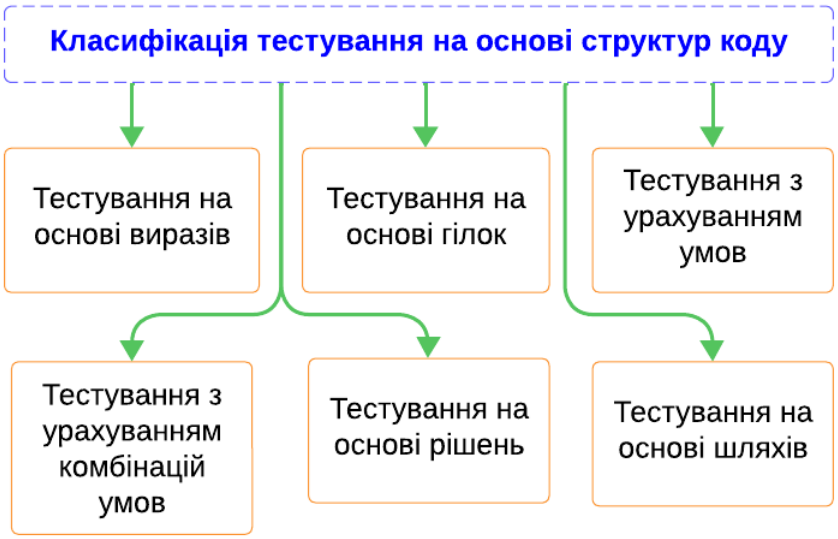
2.2.19. Класифікація тестування на основі структур коду

Тестування на основі структур коду передбачає можливість дослідження логіки виконання коду в залежності від різних ситуацій і включає.

Тестування на основі виразів – техніка тестування, в якій перевіряється коректність (і сам факт) виконання окремих виразів у коді.

Тестування на основі гілок – техніка тестування, в якій перевіряється виконання окремих гілок коду (під гілком розуміється атомарна частина коду, виконання якої відбувається або не відбувається залежно від істинності чи хибності певної умови).

Тестування з урахуванням умов — техніка тестування, у якій перевіряється виконання окремих умов (умовою вважається вираз, що може бути обчислено значення «істина» чи «брехня»).



Тестування з урахуванням комбінацій умов – техніка тестування, у якій перевіряється виконання складних (складових) умов. Тестування на основі окремих умов, що породжують розгалуження це техніка тестування, в якій перевіряється виконання таких окремих умов у складі складних умов, які визначають результат обчислення всього складного становища.

Тестування на основі рішень – техніка тестування, в якій перевіряється виконання складних розгалужень (з двома та більш можливими варіантами). Незважаючи на те, що «два варіанти» сюди також підходить, формально таку ситуацію варто віднести до тестування на основі умов.

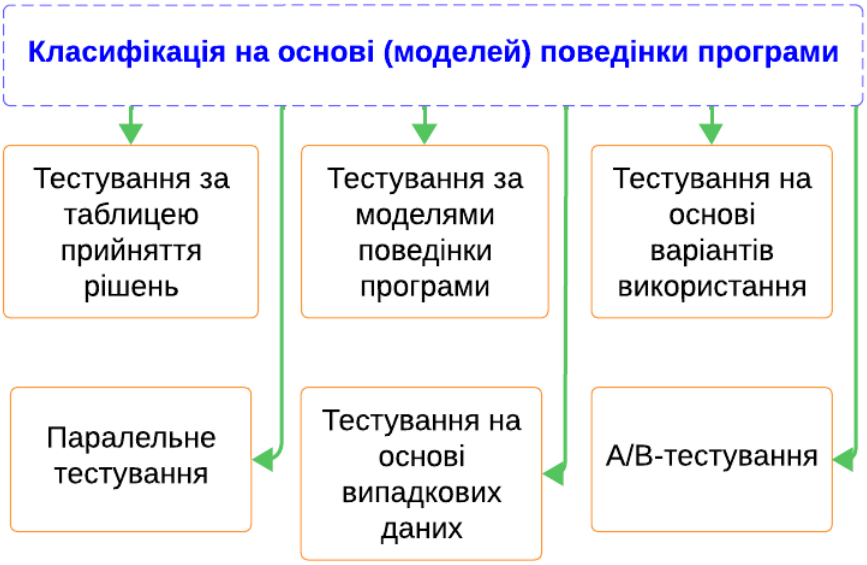
Тестування на основі шляхів – техніка тестування, в якій перевіряється виконання всіх або деяких спеціально вибраних шляхів у коді програми.

2.2.20. Класифікація на основі (моделей) поведінки програми

Тестування за таблицею прийняття рішень – техніка тестування, в якій тест-кейси розробляються на основі т.зв. таблиці прийняття рішень, в якій відображені вхідні дані (та їх комбінації) та впливу на застосунок, а також відповідні вихідні дані та реакції застосунка.

Тестування за моделями поведінки програми – техніка тестування, в якій дослідження програми (і розробка тест-кейсів) будується на якійсь моделі: таблиці прийняття рішень, таблиці або

діаграмі станів, сценаріїв користувача, моделі навантаження і т. п.



Тестування на основі варіантів використання - техніка тестування, в якій тест-кейси розробляються на основі варіантів використання. Варіанти використання виступають переважно джерелом інформації для кроків тест-кейсу, тоді як набори вхідних даних зручно розробляти за допомогою технік вибору вхідних даних. У загальному випадку джерелом інформації для розробки тест-кейсів у цій техніці можуть виступати не тільки варіанти використання, але й інші вимоги користувача в будь-якому їх вигляді. Якщо методологія розробки проекту передбачає використання історій користувача, цей вид тестування може бути замінений тестуванням на основі історій користувача.

Паралельне тестування – техніка тестування, в якій поведінка нового (або модифікованого) застосунка порівнюється з поведінкою еталонного застосунка (приблизно працюючого правильно). Термін «паралельне тестування» також може використовуватися для позначення способу проведення тестування, коли кілька тестувальників або систем автоматизації виконують одночасно роботу, тобто паралельно. Дуже рідко (і не зовсім правильно) під паралельним тестуванням розуміють мутаційне тестування .

Тестування на основі випадкових даних – техніка тестування, в якій вхідні дані, дії або навіть самі тест-кейси вибираються на основі псевдо випадкових значень так, щоб відповідати операційному профілю – підмножині дій, що відповідають певній ситуації або сценарію роботи

з застосунком.

A/B-тестування – техніка тестування, в якій досліджується вплив на результат виконання операції зміни одного з вхідних параметрів. Однак куди частіше можна зустріти трактування A/B-тестування як техніку тестування зручності використання, в якій користувачам випадковим чином пропонують різні варіанти елементів інтерфейсу, після чого оцінюється різниця в реакції користувачів.

2.2.21. Класифікація за моментом виконання (хронології)

Незважаючи на численні спроби створити єдину хронологію тестування, зроблені багатьма авторами, можна сміливо стверджувати, що загальноприйнятого рішення, яке однаково підходило б для будь-якої методології управління проектами, будь-якого окремого проекту та будь-якої його стадії, просто не існує. Якщо спробувати описати хронологію тестування однією загальною фразою, можна сказати, що відбувається поступове нарощування складності самих тест-кейсів і складності логіки їх вибору.

Загальна універсальна логіка послідовності тестування полягає в тому, щоб розпочинати дослідження кожного завдання з простих позитивних тест-кейсів, до яких поступово додавати негативні (але теж досить простих). Лише після того, як найбільш типові ситуації покриті простими тест-кейсами, слід переходити до більш складних (знов таки, починаючи з позитивних). Такий підхід не догма, але до нього варто прислухатися, так як поглиблення на початкових етапах на негативних тест-кейсах може призвести до ситуації, в якій застосунок відмінно справляється з купою неприємностей, але не працює на елементарних повсякденних завданнях.

Розглянемо послідовність тестування, побудовану за ієрархією

компонентів.

Висхідне тестування – інкрементальний підхід до інтеграційного тестування, в якому насамперед тестуються низько рівневі компоненти, після чого процес переходить на все більш високорівневі компоненти.

Східне тестування – інкрементальний підхід до інтеграційного тестування, в якому в першу чергу тестуються високорівневі компоненти, після чого процес переходить на все більш і більш низькорівневі компоненти.

Гібридне тестування – комбінація висхідного та східного тестування, що дозволяє спростити та прискорити отримання результатів оцінки програми.

2.3. Послідовність тестування, побудована за концентрацією уваги на вимогах та їх складових

1) Тестування вимог, яке може змінюватись від побіжної оцінки у стилі «чи все нам зрозуміло» до дуже формальних підходів, у разі первинне стосовно тестування те, як ці вимоги реалізовані.

2) Тестування реалізації функціональних складових вимог логічно проводити до тестування реалізації нефункціональних складових, так як якщо щось просто не працює, то перевіряти продуктивність, безпеку, зручність та інші нефункціональні складові безглуздо, а найчастіше і зовсім неможливо.

3) Тестування реалізації нефункціональних складових вимог часто стає логічним завершенням перевірки того, як реалізовано ту чи іншу вимогу.

Типові загальні сценарії використовують у тому випадку, коли немає явних передумов реалізації іншої стратегії. Такі сценарії можуть змінюватись та комбінуватися (наприклад, весь «типовий загальний сценарій 1» можна повторювати на всіх кроках «типового загального сценарію 2»).

Типовий загальний сценарій 1:

1) Димове тестування.

2) Тестування критичного шляху .

3) Розширене тестування .

Типовий загальний сценарій 2:

1) Модульне тестування .
2) Інтеграційне тестування .
3) Системне тестування .

Типовий загальний сценарій 3:
1) Альфа-тестування .
2) Бета-тестування .
3) Гамма-тестування.

На завершення ще раз наголосимо, що розглянуті тут класифікації тестування не є чимось канонічним та непорушним. Вони лише покликані впорядкувати величезний обсяг інформації про різні види діяльності тестувальників та спростити запам'ятовування відповідних фактів.

2.3. Альтернативні та додаткові класифікації тестування

Для повноти картини залишається лише показати альтернативні погляди класифікацію тестування. У наступній класифікації зустрічаються пункти які раніше не розглядалися (рис. 2.2).

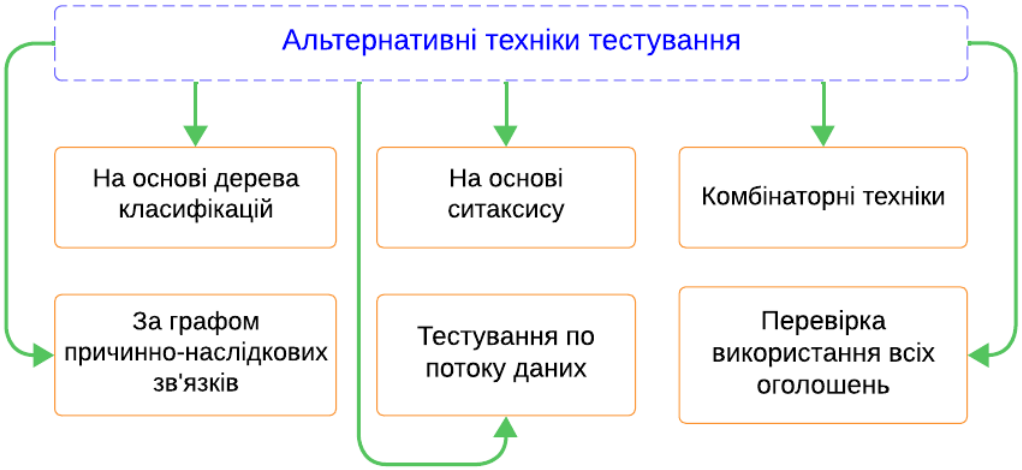


Рисунок 2.2 – Класифікація тестування згідно з ISO/IEC/IEEE 29119-4

Тестування на основі дерева класифікацій – техніка тестування, в якій тест-кейси створюються на основі ієрархічно організованих наборів еквівалентних вхідних та вихідних даних.

Тестування на основі синтаксису – техніка тестування, в якій тест-кейси створюються на основі визначення наборів вхідних та вихідних даних.

Комбінаторні техніки або комбінаторне тестування – спосіб вибрати відповідний набір комбінацій тестових даних для досягнення встановленого рівня тестового покриття у разі, коли перевірка всіх можливих наборів значень тестових даних неможлива за наявний час. Існують такі комбінаторні техніки:

- *тестування всіх комбінацій* – тестування всіх можливих комбінацій усіх значень усіх тестових даних (наприклад, усіх параметрів функції).
- *тестування з вибором значень-представників* – тестування, при якому за одному значенню з кожного набору тестових даних має бути використано хоч би в одному тест-кейсі.
- *тестування з вибором базового набору значень* – тестування, при якому виділяється набір значень (базовий набір), який використовується для проведення тестування в першу чергу, а далі тест-кейси будуються на основі вибору всіх базових значень, крім одного, яке замінюється значенням, що не входить базовий набір.

Тестування за графом причинно-наслідкових зв'язків – техніка тестування, в якій тест-кейси розробляються на основі графа причинно-наслідкових зв'язків (графічного представлення вхідних даних та впливів із пов'язаними з ними вихідними даними та ефектами).

Тестування по потоку даних – сімейство технік тестування, заснованих на виборі окремих шляхів із потоку управління з метою дослідження подій, пов'язаних із зміною стану змінних. Ці техніки дозволяють виявити такі ситуації, як:

- змінна визначена, але ніде не використовується;
- змінна використовується, але не визначена;
- змінна визначена кілька разів перед тим, як вона використовується;
- змінну видалено до останнього випадку використання.

Перевірка використання всіх оголошень – тестовий набір перевіряє, що для кожної змінної існує шлях від її визначення до використання у обчисленнях або умовах.