**Beet Seed — відпрацюй навички на базовому рівні.**

**1. Склади порівняльну таблицю функціонального, нефункціонального і пов’язаного зі змінами видів тестування.**

**Порівняння має містити такі блоки:**

**що перевіряється;**

**коли застосовується;**

**обмеження;**

**особливості**

| **Вид тестування** | **Що перевіряється** | **Коли застосовується** | **Обмеження** | **Особливості** |
| --- | --- | --- | --- | --- |
| **Функціональне** | Задані функції та поведінка системи відповідно до специфікацій: бізнес-логіка, GUI (розмір, шрифт, колір, послідовність), взаємодія компонентів. | На різних рівнях (модульне, інтеграційне, системне, приймальне); після розробки функціоналу | Не оцінює продуктивність, безпеку, зручність; обмежене рамками вимог. | Базується на специфікаціях; підходи: тестування за вимогами, за бізнес-процесами. |
| **Нефункціональне** | Характеристики якості: продуктивність (навантаження, стрес, стабільність, об’єм), встановлення, юзабіліті, безпека, відмовостійкість, сумісність, конфігурації. | Зазвичай після функціональних перевірок, на етапах системного тестування чи перед релізом. | Потребує спеціальних інструментів та середовищ; залежить від умов виконання (мережа, дані, обладнання). | Вимірює не що робить система, а як вона це робить; часто визначає готовність до випуску. |
| **Димове тестування** | Чи запускається збірка та виконує основні критичні функції. | Після збирання нової або оновленої версії продукту. | Не перевіряє повністю функціонал; може пропустити дрібні дефекти. | Швидка базова перевірка «життєздатності» збірки. |
| **Регресійне тестування** | Чи збереглась працездатність старих функцій після змін (виправлення багів, оновлення середовища, бази, серверних налаштувань). | Після будь-яких змін у коді чи середовищі. | Витратне за часом і ресурсами без автоматизації. | Може включати як функціональні, так і нефункціональні тести; ключове для стабільності продукту. |
| **Повторне тестування** | Чи виправлено конкретний дефект та чи працює змінений функціонал. | Після виправлення багів. | Перевіряє лише виправлені місця; не гарантує стабільність решти системи. | Виконується на основі тих самих тест-кейсів, де раніше були баги. |
| **Тестування збірки** | Чи відповідає версія критеріям якості для початку детального тестування. | Одразу після отримання нової збірки, перед запуском основних тестів. | За суттю дублює smoke testing, але з акцентом на формальні критерії якості. | Використовується як «фільтр» для відкидання неякісних збірок. |
| **Санітарне тестування** | Чи працює конкретна функція/невеликий набір функцій після внесених змін. | При отриманні збірки з локальними виправленнями, без повного прогона regression. | Обмежене вузьким обсягом перевірки. | Є частиною regression, часто виконується вручну для швидкої перевірки. |

**2. Поясни, в чому різниця між регресією та ретестингом (5 речень).**

Різниця між регресією і ретестингом у тому, що вони перевіряють різні речі.

Ретестинг робиться тоді, коли баг уже виправили - ми запускаємо ті ж самі тести, щоб переконатися, що помилка справді зникла.

Регресійне тестування охоплює ширший спектр і перевіряє, чи після виправлення або інших змін у коді не зламалася вже існуюча функціональність.

Тобто ретест - це точковий контроль конкретної проблеми, а регресія- глобальна перевірка стабільності системи.

Обидва підходи зазвичай виконуються разом, щоб і баг виправити, і впевнитися, що нічого нового не зламалося.

**Beet Sprout — детальніше заглибся в практику.**

* **1. Виконай завдання попереднього рівня.**

**2. Як ти вважаєш, чи можливе для продукту проведення тільки функціонального тестування, без перевірки нефункціональних вимог?**

Якщо так – в яких випадках?

Якщо ні – чому?

Обґрунтуй свою відповідь.

Так, провести тільки функціональне тестування можливо, але це виправдано лише в окремих випадках. Наприклад, коли створюється прототип, MVP або внутрішній інструмент із дуже простими завданнями, достатньо перевірити лише правильність роботи функцій. У таких проєктах нефункціональні вимоги (продуктивність, безпека, зручність тощо) можуть не мати вирішального значення.

Проте для більшості продуктів обмежитися лише функціональним тестуванням неможливо. Система може виконувати всі заявлені дії, але при цьому бути повільною, нестабільною або небезпечною для користувачів. Це означає, що без нефункціонального тестування продукт не можна вважати готовим до використання.

Отже, функціональне тестування без нефункціонального можливе лише для простих і допоміжних рішень. У всіх інших випадках потрібна комплексна перевірка, щоб гарантувати якість і надійність програмного забезпечення.

**3. Як ти розумієш необхідність проведення smoke (димового) тестування? Чи завжди воно є доречним?**

Smoke-тестування є необхідним, адже воно дозволяє на початковому етапі перевірити життєздатність збірки: чи запускається система та чи працюють всі її ключові функції. Це своєрідний «фільтр», який дає змогу відсіяти нестабільні версії ще до проведення повного циклу тестування, тим самим економлячи час і ресурси команди.

Його проведення можна вважати доречним у більшості випадків, особливо за умов активної розробки та частих оновлень продукту. Проте в дуже малих або простих проєктах, де ризики мінімальні, smoke-тестування може бути не таким обов’язковим.

Можна підсумувати: попри те, що не завжди існує критична необхідність у його застосуванні, для комерційних і масштабних продуктів smoke-тестування є важливою практикою забезпечення якості.