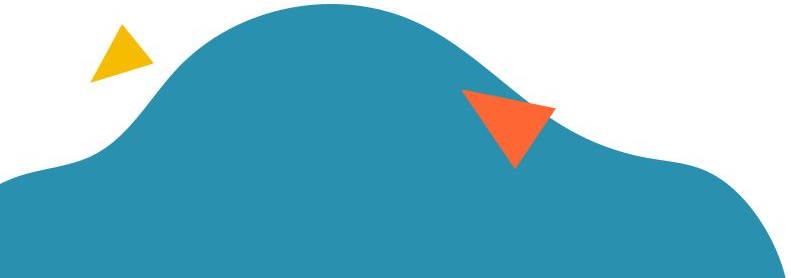
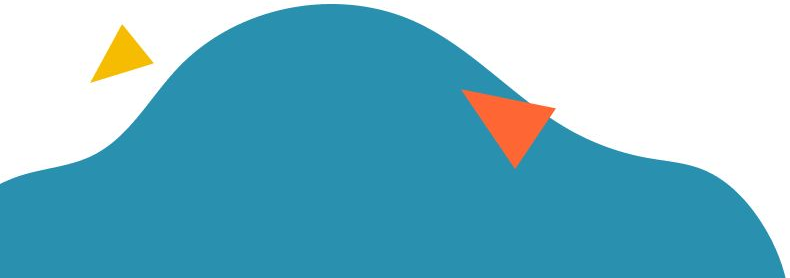
7 ПРИНЦИПІВ ТЕСТУВАННЯ, РІВНІ ТЕСТУВАННЯ, ВИДИ ТЕСТУВАННЯ

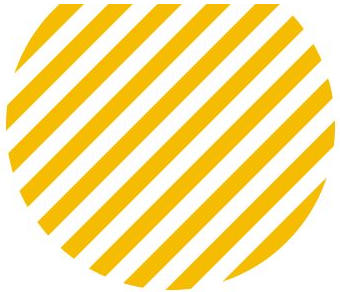


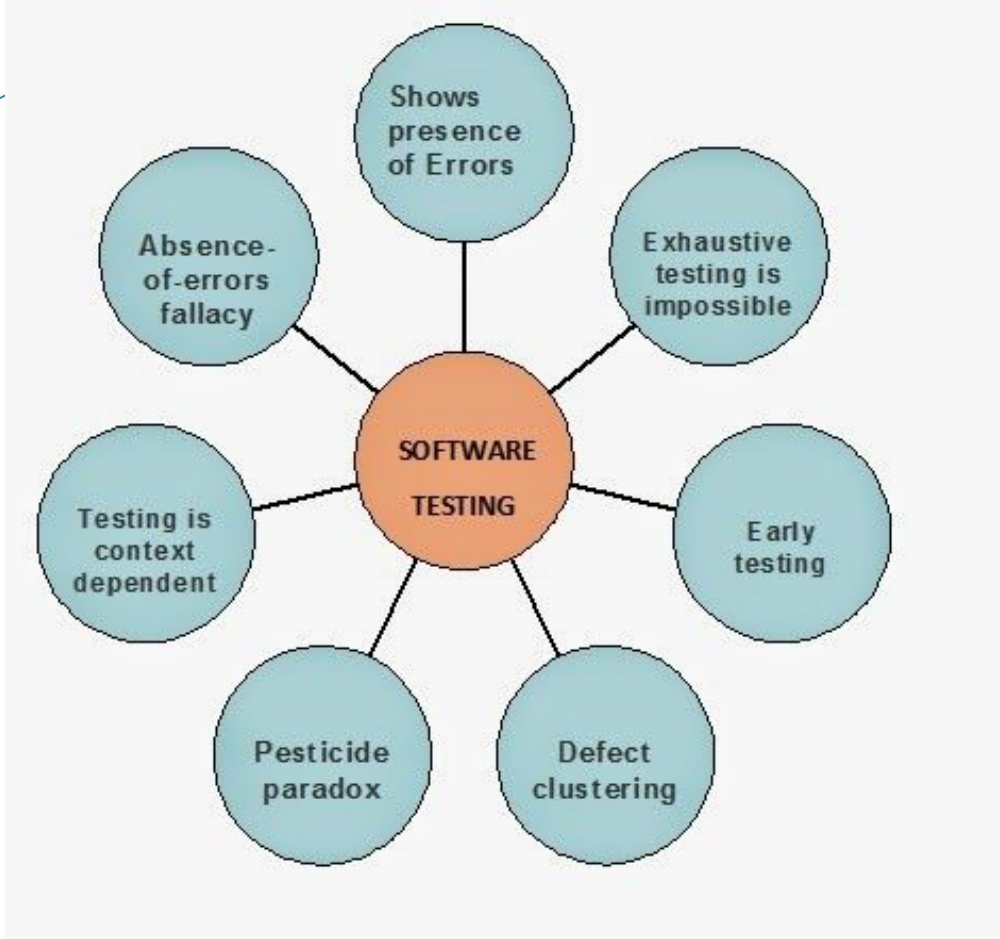
Що будемо вивчати?

1. 7 принципів тестування
2. Психологія тестування
3. Рівні тестування
4. Види тестування

7 принципів тестування






Shows presence of Errors
Absence-of-errors fallacy
Exhaustive testing is impossible
SOFTWARE TESTING
Testing is context dependent
Early testing
Pesticide paradox
Defect clustering

1. Тестування демонструє наявність дефектів, а не їх відсутність

Тестування може показати, що дефекти присутні, але може довести, що їх немає. Мета тестування - знизити ймовірність наявності дефектів, що у програмному забезпеченні, але, навіть якщо дефекти були виявлено, тестування не доводить повну коректність роботи ПО.

Довести, що чогось немає, у принципі, важко. Скільки б білих лебедів ми не бачили, це ще не є підставою стверджувати, що «всі лебеді білі». Але якщо ми побачимо хоча б одного чорного лебедя, то може сказати, що «не всі лебеді білі».

1. Тестування демонструє наявність дефектів, а не їх відсутність

Скільки б успішних тестів ви не провели, ви не можете стверджувати, що немає таких тестів, які б не знайшли помилку. Але якщо ми знайшли хоча б один дефект, ми вже можемо стверджувати, що в цьому ПЗ є дефекти.

Втім, це аж ніяк не означає, що тестування марне і не може підвищити наш рівень впевненості як ПЗ. Тестування зменшує ймовірність невиявлених дефектів, які залишаються в ПЗ, але завжди слід пам'ятати, що навіть якщо дефекти не знайдені, це ще не є доказом того, що їх немає.

(Деталіньше про"Теорії чорного лебедя" ви можете прочитати - http://design-for.net/page/teorija-chernogo-lebedja)

2. Вичерпне тестування неможливе

Повне тестування з допомогою всіх комбінацій вводів і передумов фізично нездійсненно (крім очевидних випадків) і ефективно. Замість спроб «протестувати все» нам потрібен підхід до тестування (стратегія), який забезпечить правильний обсяг тестування для даного проекту, даних замовників (та інших зацікавлених осіб) та даного продукту. При визначенні, який обсяг тестування достатній, необхідно враховувати рівень ризику, включаючи технічні ризики та ризики, пов'язані з бізнесом, та обмеження проекту як час і бюджет.

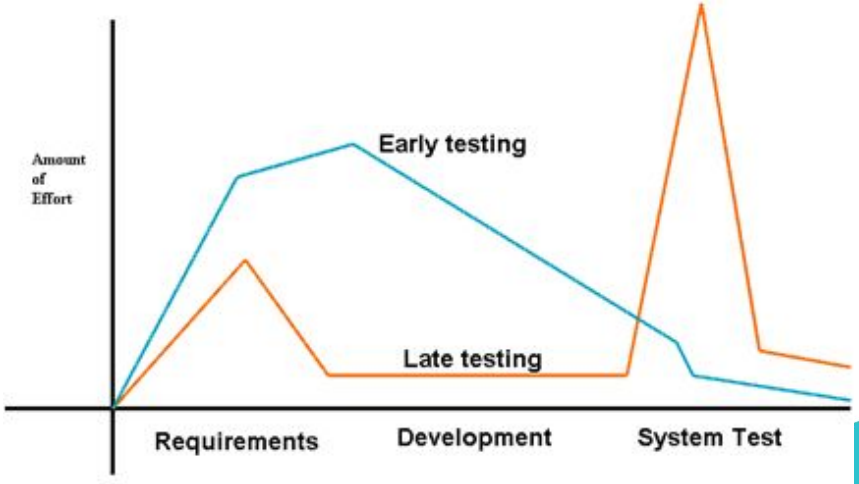
- Risk analysis: - загальний процес виявлення та оцінки ризиків.
- Exhaustive testing: - підхід до тестування, при якому набір тестів включає всі комбінації вхідних значень і попередніх умов.

3. Раннє тестування (зберігає час та гроші)

Для знаходження дефектів на ранніх стадіях як статичні, так і динамічні активності з тестування повинні бути розпочаті якомога раніше в життєвому циклі розробки програмного забезпечення.

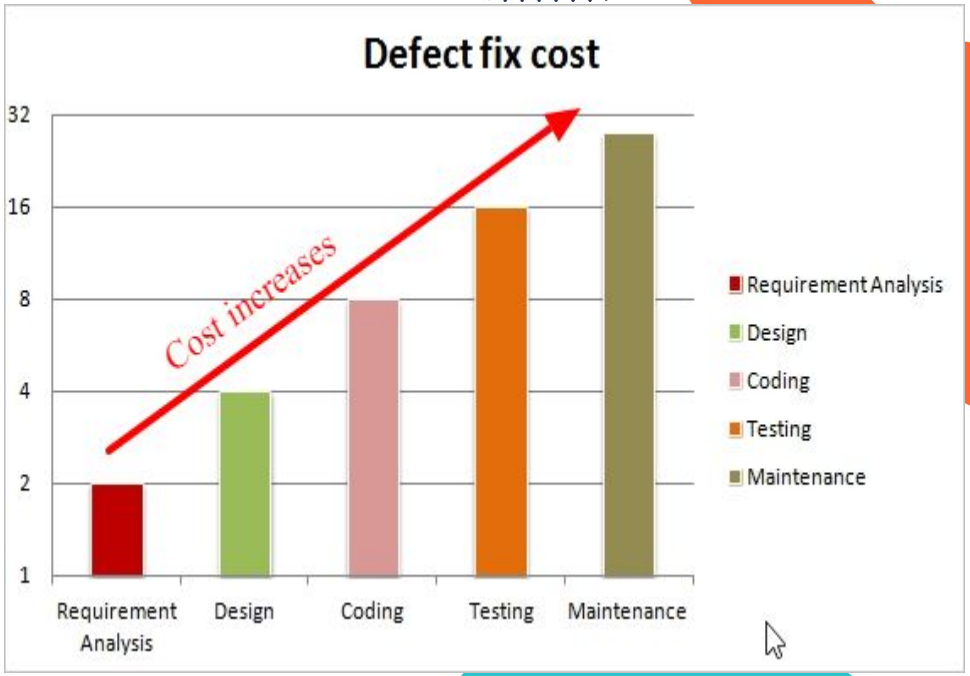
Static testing: Тестування робочого продукту без виконання коду.

Dynamic testing: тестування, яке включає виконання програмного забезпечення компонента чи системи.



"Вартість" дефекта в ПЗ

- Дефект, знайдений на етапі збору та узгодження вимог обходиться найдешевше. На цій стадії розробки продукту ще не почалася і змінити/доповнити/видалити вимогу не складе труднощів.
- Якщо дефект виявлено на етапі розробки архітектурного рішення, виправити його буде складніше: потрібно повернутися на попередню фазу та виправити вимоги, провести impact analysis (перевірити, що зміна не вплине на суміжні області) і лише після продовжити розробку.
- Якщо ж дефект, внесений ще на рівні вимог, «дожив» до етапу системного або приймального тестування, його виправлення буде дуже дорогим – адже доведеться внести зміни не тільки в код, але, можливо, і в архітектуру, і в вимоги. Іноді дефекти, виявлені на пізній стадії, взагалі не виправляються, тому що вартість виправлення дуже дорога.



4. Кластеризація дефектів

Кластеризація дефектів при тестуванні програмного забезпечення означає, що більшість дефектів містяться в декількох модулях, тобто щільність накопичення дефектів у різних елементах програми може відрізнятися.

Кластеризація дефектів у тестуванні програмного забезпечення заснована на принципі Парето, також відомому як правило 80-20 де зазначено, що приблизно 80% проблем викликані 20% модулів.

Про те, де шукати кластери дефектів, можна дізнатися ще на ранніх етапах, коли проводиться статичне тестування (наприклад, code review та аналіз коду за допомогою спеціальних інструментів). Коли дійде до динамічного тестування, можна сфокусуватися на тих областях, де було виявлено більше дефектів статичними методами.

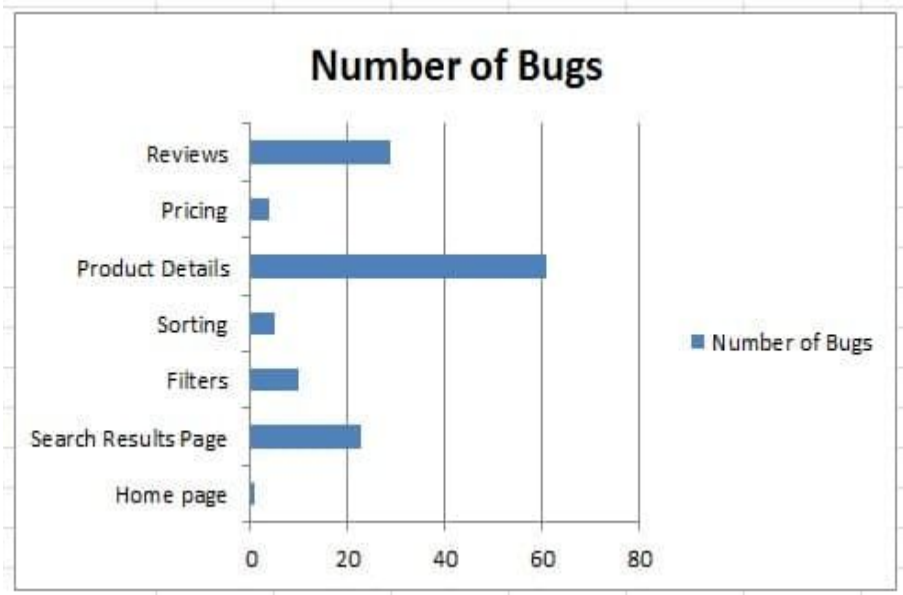
Також корисно проводити аналіз першопричин (root cause analysis), щоб запобігти повторній появі дефектів, виявити причини виникнення скупчень дефектів і спрогнозувати потенційні скупчення дефектів у майбутньому.

Root cause analysis - методика аналізу, спрямовану виявлення корінних причин дефектів. Спрямовуючи коригувальні заходи на причини дефектів ми скорочуємо їх можливість повторення в майбутньому.

4. Кластеризація дефектів

На зображенні показано, що найбільше дефектів містять модулі - "Product details", "Reviews" і "Search Result Page".
Причини, чому відбувається кластеризація дефектів:

- бракує часу
- нестача ресурсів
- складна логіка
- мінливі вимоги
- недостатній рівень кваліфікації спеціалістів



5. Парадокс пестициду

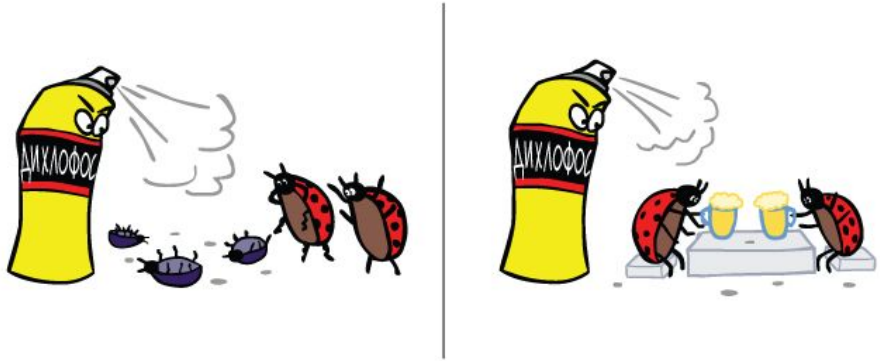
Якщо ті самі тести будуть виконуватися знову і знову, зрештою ці тести більше не будуть знаходити нових дефектів.

Що потрібно робити для підвищення ефективності тест-кейсу та виявлення нових багів:

* модифікувати існуючі тест-кейси

* змінювати тестові дані

* створювати нові тест-кейси



6. Тестування залежить від контексту

Тестування виконується по-різному, залежно від контексту. Наприклад, програмне забезпечення управління виробництвом, в якому критично важлива безпека (**safety-critical soft**), тестується інакше, ніж тестування сайту інтернет-магазину.

7. Оманливість про відсутність помилок

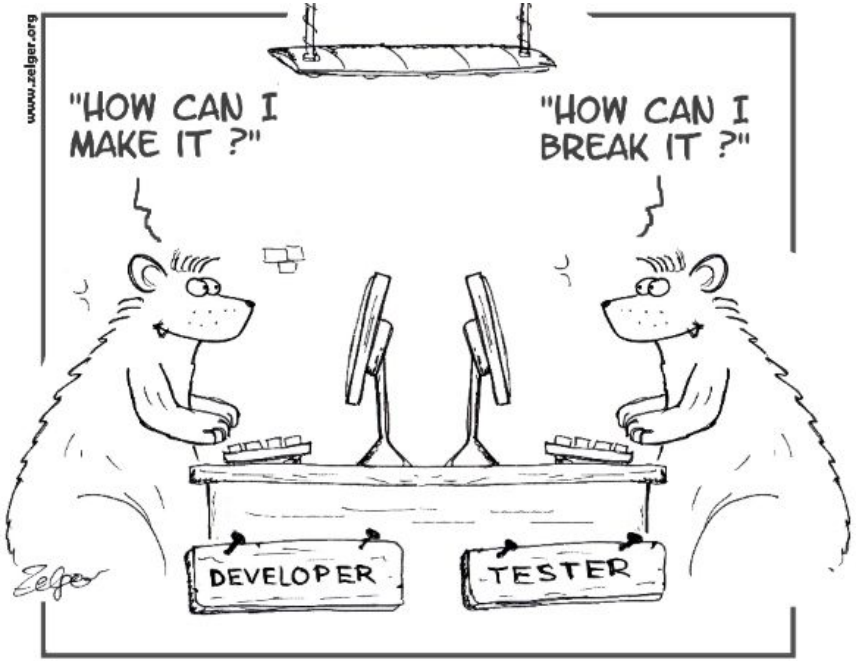
Навіть ретельне тестування всіх зазначених вимог та виправлення всіх виявлених дефектів може призвести до створення системи, яка буде:

- незручною у використанні;
- не відповідатиме очікуванням користувачів;
- буде гірше, порівняно з іншими конкуруючими системами.

Психологія тестування

Мислення розробника і мислення тестувальника

ПСИХОЛОГІЯ ТЕСТУВАННЯ



Психологія тестування

Спосіб мислення тестувальника повинен включати:

- цікавість
- професійний песимізм
- критичний погляд
- увага до деталей

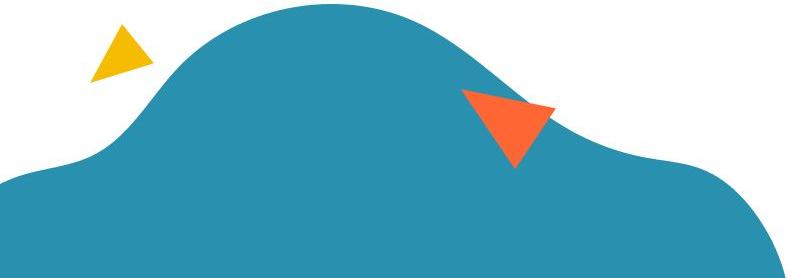
Психологія тестування

Способи гарного спілкування:

- Почніть із співпраці, а не з битв;
- Щоб уникнути нерозуміння, уточніть, що інша людина зрозуміла, що було сказано.
- Повідомляйте результати тестування нейтрально, фокусуйтесь лише на фактах;
- Не критикуйте людину, яка створила дефект

Test levels

Рівні тестування



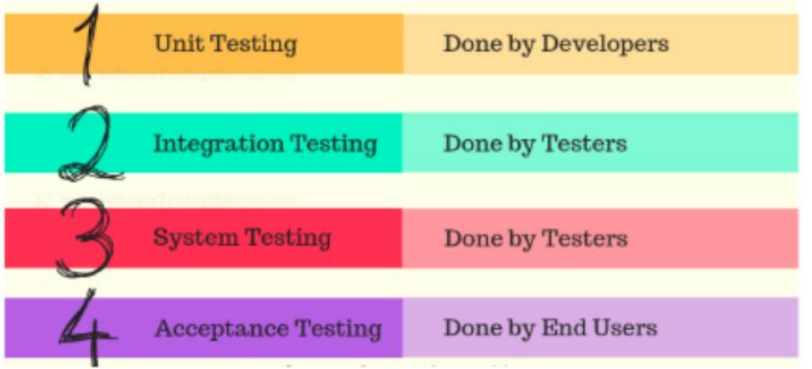
Рівні тестування

Модульне тестування (Unit testing)

Інтеграційне тестування (Integration testing)

Системне тестування (System testing)

Приймальне тестування (Acceptance testing)



ПЗ і модулі

Програмний продукт

Модулі

Модуль

- частина програмного коду, що виконує одну функцію з погляду функціональних вимог;
- програмний модуль - мінімальний компілюваний елемент програмної системи;
- завдання у списку проектних завдань з погляду менеджера;
- клас чи безліч класів із єдиним інтерфейсом;

Модульне (unit) тестування

Перевірка коректності програми на рівні мінімально можливого для тестування компонента, наприклад окремого класу, модуля або функції.

Зазвичай модульне тестування проводиться викликом коду, який необхідно перевірити і за підтримки середовищ розробки різних фреймворків для модульного тестування або засобів налагодження.

Цілі модульного тестування

- достовірність відповідно до вимог кожного окремого модуля до його інтеграції до складу системи;
- одержання працездатного коду з найменшими витратами;
- визначення ступеня готовності системи до переходу на наступний рівень розробки та тестування;
- розробка тестів, які перевіряють роботу кожної нетривіальної функції чи методу модуля.

Переваги і недоліки

+плюси

- легкість виявлення помилок
- спрощення інтеграції
- документування коду
- усунення сумнівів щодо надійності окремих модулів

-мінуси

- збільшення терміну розробки
- при зміні вимог, створені раніше тести стають неактуальними

Інтеграційне тестування

Модульна інтеграція

Система інтеграція

Модульне інтеграційне тестування

Тестування частини системи, що складається із двох і більше модулів (компонентів).

Основне завдання - пошук дефектів, пов'язаних з помилками у реалізації та інтерпретації взаємодії між модулями.

А що зі мною?

Системний інтеграційний рівень

Методи інтеграційного тестуваннння

- **висхідне тестування -** спочатку проводиться модульне тестування, та був поетапне тестування інтеграції.
- **монолітне тестування –** тестування інтеграції без попереднього модульного тестування.
- **низхідне тестування -** тестується верхній керуючий рівень системи, без модулів низького рівня. Потім поступово з більш високорівневими модулями інтегруються більш низькорівневі.

Системне тестування

Основне завдання системного тестування - це перевірка функціональних та нефункціональних вимог у системі загалом, виявлення дефектів:

- відсутня чи неправильна функціональність;
- неправильне використання ресурсів системи;
- непередбачені комбінації даних від користувача;
- несумісність із оточенням;
- непередбачені сценарії використання;
- незручність у використанні
- і багато іншого

Підходи до системного тестування

- **на базі вимог** - для кожної вимоги пишуться тестові випадки, що перевіряють виконання цієї вимоги
- **на базі випадків використання (use case)** - на основі уявлення про засоби використання продукту створюються випадки використання системи. За конкретною нагодою використання можна визначити один або більше сценаріїв, на кожен сценарій створюються тестові випадки, які потім виконуються

Приймальне тестування

Формальний процес тестування, який перевіряє відповідність системи вимогам та проводиться з метою:

- визначення - чи задовольняє система приймальним критеріям;
- винесення рішення замовником або іншою відповідальною особою - приймається додаток чи ні.

Приймальний тест проводиться на підставі набору типових тестових випадків і сценаріїв, розроблених на підставі вимог до цього додатка.

Фаза приймального тестування триває доти, доки замовник не виносить рішення про відправлення програми на доопрацювання або випуск програми.

Альфа-тестування *(alpha testing)* – це вид приймального тестування, яке зазвичай проводиться пізньої стадії розробки товару і включає імітацію реального використання товару штатними розробниками чи командою тестувальників. Зазвичай альфа тестування полягає в систематичній перевірці всіх функцій програми з використанням технік тестування «білої скриньки» та «чорної скриньки».

Переваги альфа-тестування:

- Забезпечує найкраще уявлення про надійність програмного забезпечення на ранній стадії.
- Допомагає моделювати поведінку користувача та навколишнє середовище у режимі реального часу.
- Виявляє багато серйозних помилок.
- Дає можливість раннього виявлення помилок щодо дизайну та функціональності.

Недоліки альфа-тестування:

- Функціональність не може бути перевірена на всю глибину, оскільки програмне забезпечення все ще перебуває на стадії розробки. Іноді розробники та тестувальники незадоволені результатами альфа-тестування.

Бета-тестування*(beta testing)* – Інтенсивне використання майже готової версії продукту з метою виявлення максимальної кількості помилок у його роботі для їх подальшого усунення перед остаточним виходом (релізом) продукту на ринок до масового споживача. Бета-тестування є реально працюючою версією програми з повним функціоналом. І завдання бета-тестів – оцінити можливості та стабільність роботи програми з погляду її майбутніх користувачів.

На відміну від альфа-тестування, що проводиться силами штатних розробників або тестувальників, бета-тестування передбачає залучення добровольців з-поміж звичайних майбутніх користувачів продукту, яким доступна згадана попередня версія продукту (так звана бета-версія).

Бета-тестування може бути:

- *Закритим: Програма тестується в невеликій групі користувачів на запрошення.*
- *Відкритим: Цей варіант дозволяє протестувати програму у більшій групі та отримати великий обсяг зворотного зв'язку. Будь-який користувач зможе приєднатися до відкритого бета-тестування та надіслати особистий відгук.*

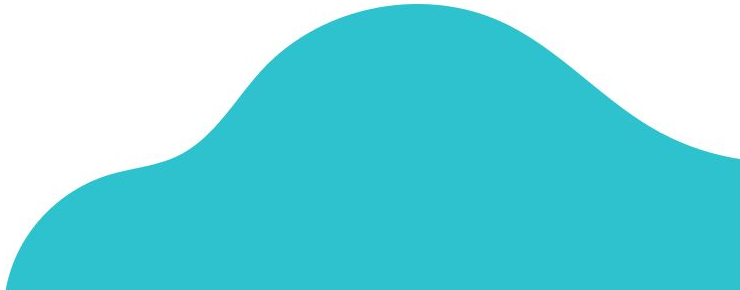
Переваги бета-тестування:

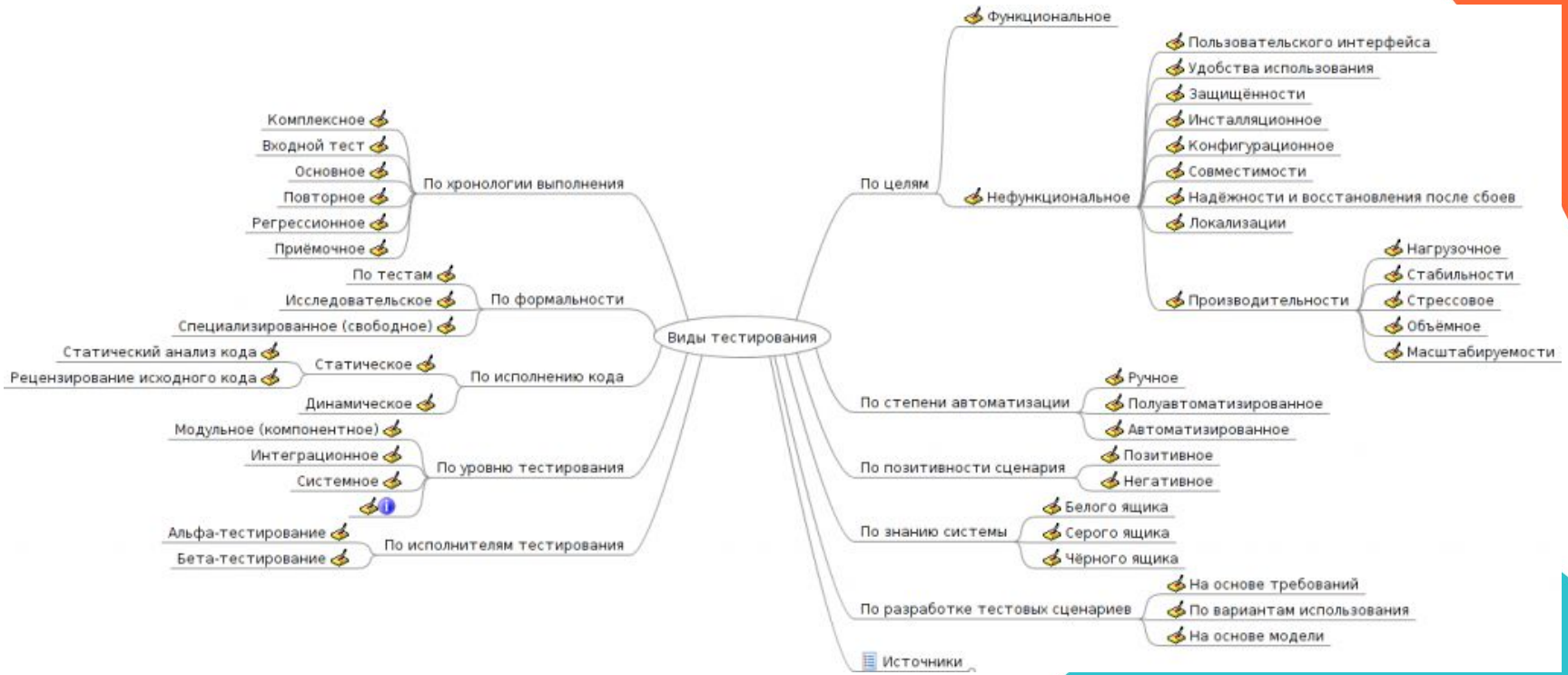
- Знижує ризик виходу продукту з ладу у вигляді валідації клієнта.
- Бета тестування дозволяє компанії тестувати інфраструктуру після запуску.
- Підвищує якість продукції завдяки зворотному зв'язку із клієнтами.
- Є економічним способом збору даних проти аналогічними методами.
- Створює доброзичливість із клієнтами та підвищує задоволеність клієнтів.

Недоліки бета-тестування:

- Управління тестуванням – проблема. У порівнянні з іншими типами тестування, які зазвичай виконуються всередині компанії в контрольованому середовищі, бета-тестування виконується в реальному світі, де компанія рідко має контроль.
- Пошук правильних користувачів бета-версії та підтримка їхньої участі може викликати труднощі.

Види тестування




Виды тестирования
По целям
Функциональное
Нефункциональное
Пользовательского интерфейса
Удобства использования
Защищённости
Инсталляционное
Конфигурационное
Совместимости
Надёжности и восстановления после сбоев
Локализации
Производительности
Нагрузочное
Стабильности
Стрессовое
Объёмное
Масштабируемости
По степени автоматизации
Ручное
Полуавтоматизированное
Автоматизированное
По позитивности сценария
Позитивное
Негативное
По знанию системы
Белого ящика
Серого ящика
Чёрного ящика
По разработке тестовых сценариев
На основе требований
По вариантам использования
На основе модели
Источники
По хронологии выполнения
Комплексное
Входной тест
Основное
Повторное
Регрессионное
Приёмочное
По формальности
По тестам
Исследовательское
Специализированное (свободное)
По исполнению кода
Статическое
Статический анализ кода
Рецензирование исходного кода
Динамическое
По уровню тестирования
Модульное (компонентное)
Интеграционное
Системное
По исполнителям тестирования
Альфа-тестирование
Бета-тестирование

Види тестування

1. За цілями: функціональне та нефункціональне
2. Пов'язане із змінами: smoke, re-test, regression, build verification, sanity
3. Структурне: строкове покриття, покриття шляху, покриття рішення, покриття умов
4. За ступенем автоматизації: ручне, автоматизоване
5. За позитивністю сценарію: позитивне, негативне
6. За доступом до коду програмного продукту: "білої скриньки", "чорної скриньки", "сірої скриньки"
7. За рівнем
8. За виконавцем: альфа та бета

Функціональне тестування

Функціональне тестування системи включає тести з оцінки функцій, які має виконувати система. Функціональні вимоги можуть бути описані в робочих продуктах (вимоги, специфікація, бізнес-потреба, історія користувача, сценарій використання) або у функціональній специфікації, а можуть бути взагалі не задокументовані. Функції системи дають у відповідь питання «що робить система».

Функціональне тестування

Направлено на тестування всіх функцій системи, щоб підтвердити, що кожна функція програми працює відповідно до документації.

Елементи функціонального тестування:

- підготовка тестових даних на основі описаної документації;
- бізнес-вимоги як частина функціонального тестування;
- одержання результатів на основі специфікації;
- проходження тест-кейсів;
- аналіз фактичних та очікуваних результатів.

Функціональне тестування може бути проведене як у суворій відповідності до літери специфікації, так і на основі бізнес-процесу (тобто відповідно до знань системи).

Переваги функціонального тестування:

- у межах тестування ми «копіюємо» безпосереднє використання системи;
- Тестування, як правило, проводиться в умовах близьких до реальних.

Недоліки:

- існує можливість пропустити кілька помилок логіки ПЗ під час перевірки функціоналу програми.

Нефункціональне тестування

Нефункціональне тестування системи виконується для оцінки таких характеристик системи та програмного забезпечення, як зручність використання, продуктивність чи безпека. За класифікацією параметрів якості програмного забезпечення слід звернутися до стандарту ISO (ISO/IEC 25010). Нефункціональне тестування – це перевірка того, наскільки добре працює система.

Попри загальну помилку, дисфункція може і, найчастіше, повинна виконуватися на всіх рівнях тестування, і якомога раніше. Несвоєчасне виявлення дисфункційних дефектів може бути загрозою успіху всього проекту.

Підвиди нефункціонального тестування

Тестування продуктивності – Performance testing – перевірка швидкості роботи програмного забезпечення або його окремих функцій.
Тестування безпеки – Security testing – проводиться для відповіді на запитання «Чи є програма безпечною/захищеною чи ні?».
Юзабіліті тестування – Usability testing – дослідження визначення зручності використання ПО.
UI testing (інтерфейсу користувача) – перевірка програми на відповідність вимогам до графічного інтерфейсу.
Перевірка переносимості (портируємості) – Portability testing – тестування доступності перенесення окремого компонента або всього програмного забезпечення з одного оточення на інше (Windows 8.1 -> Windows 10, Windows -> MacOS).
Cross-browser testing – виконується для перевірки правильної роботи програми або системи у різних конфігураціях браузера: Mozilla Firefox, Google Chrome, Internet Explorer та Opera.
Cross-platform testing – виконується для оцінки роботи програми у різних операційних системах.

Підвиди нефункціонального тестування

Тестування установки - Installation testing - перевірка успішної установки, впровадження оновлень та видалення програм.

Тестування сумісності - Compatibility testing - тестування системи під час роботи у різних оточеннях: «залізо», софт частина тощо.

Тестування швидкості відновлення - Recovery testing - проводиться з метою визначення швидкості відновлення системи у разі софтверного крешу (крешу програмного забезпечення) або помилки заліза.

Тестування локалізації - Localization testing - перевірка ПЗ на відповідність мовних, культурних та/або релігійних норм. Локалізація – перевірка відображення всіх текстів, що переведені в ПЗ.

Навантажне тестування

№	Вид тестирования	Вид тестирования по английский	Вопрос на который отвечает тестирование
1	Нагрузочное тестирование	Load Testing[2]	Достаточно ли быстро работает система?
2	Тестирование стабильности	Stability Testing[3]	Достаточно ли надежно работает система на долгом интервале времени?
3	Тестирование отказоустойчивости	Failover Testing[4]	Сможет ли система переместиться сама на другой сервер в случае сбоя основного сервера?
4	Тестирование восстановления	Recovery Testing[5]	Как быстро восстановится система?
5	Стрессовое тестирование	Stress Testing[6]	Что произойдет при незапланированной нагрузке?
6	Тестирование объемов	Volume Testing[7]	Как будет работать система, если объем базы данных увечится в 100 раз?
7	Тестирование масштабируемости	Scalability Testing[8]	Как будет увеличиться нагрузка на компоненты системы при увеличении числа пользователей?
8	Тестирование потенциальных возможностей	Capacity Testing[9]	Какое количество пользователей может работать?
9	Конфигурационное тестирование	Configuration Testing[10]	Как заставить систему работать быстрее?
10	Тестирование сравнения	Compare Testing[11]	Какое оборудование и ПО выбрать?

Тестування безпеки

Тестування безпеки – комплекс досліджень програмного продукту, спрямований на тестування, виявлення та виправлення дефектів, пов'язаних із збереженням даних користувача, а саме:

Цілісність. Обмеження кола користувачів, які мають доступ до даних, визначення ступеня шкоди, завданої у разі втрати тих чи інших даних.
Доступність. Це вимоги про те, що ресурси повинні бути доступні авторизованому користувачеві, внутрішньому об'єкту або пристрою. Як правило, чим критичніший ресурс тим вище рівень доступності має бути.
Конфіденційність. Приховування певних ресурсів чи інформації. Під конфіденційністю можна розуміти обмеження доступу до ресурсу певної категорії користувачів, або іншими словами, за яких умов користувач має доступ до даного ресурсу.

У ході тестування, найчастіше тестувальник грає роль зломщика, і починає маніпулювати по-різному додатком:

- Спроби дізнатись пароль за допомогою зовнішніх засобів.
- Атака системи за допомогою спеціальних утиліт, що аналізують захист.
- Придушення, приголомшення системи (в надії, що вона відмовиться обслуговувати інших клієнтів).
- Цілеспрямоване введення помилок, сподіваючись проникнути в систему під час відновлення.
- Перегляд нетаємних даних, сподіваючись знайти ключ для входу в систему.

Тестування пов'язане зі змінами

smoke testing – розглядається як короткий цикл тестів, що виконується для підтвердження того, що після складання коду (нового або виправленого) додаток, що встановлюється, стартує і виконує основні функції.

retesting – проводиться для підтвердження виправлення помилки та роботи даного функціоналу.

regression testing – проводиться з метою перевірки працездатності існуючого функціоналу та відсутності сторонніх помилок після внесення поправок чи доповнень до системи.

build verification – спрямовано визначення відповідності, випущеної версії, критеріям якості початку тестування. За своїми цілями є аналогом smoke testing, спрямованого на прийняття нової версії в подальше тестування або експлуатацію.

sanity – використовується щоразу, коли ми отримуємо відносно стабільний білд, щоб визначити працездатність в деталях. Іншими словами, тут відбувається валідація того, що важливі частини функціональності системи працюють згідно з вимогами на низькому рівні.

Retesting vs regression

Регресійне тестування виконується лише при додаванні нової фічі (додаткова функціональність ПЗ) або суттєвій зміні функціоналу в системі.

Регрес можна проводити паралельно із повторним тестуванням.

Тест-кейси можуть бути автоматизовані.

В рамках регресійного тестування тест-кейси, які були відзначені раніше як Passed, повинні бути перевірені повторно.

Ретест виконується в тому ж оточенні і з тими самими даними, але на новому білді.

Повторне тестування має вищий пріоритет і має бути виконане до регресійного.

Тест-кейси не можуть бути автоматизовані.

В рамках повторного тестування (ретесту) перевіряються тільки провалені (зі статусом Failed) тест-кейси.

Retesting vs regression

Повторне тестування (Retesting)

Переваги:

- *підтверджує виправлення помилки та коректну роботу функціоналу;*
- *підвищує загальну якість продукту;*
- *потребує менше часу на верифікацію;*
- *не вимагає нових налаштувань середовища тестування.*

Недоліки:

- *потребує нового білда для верифікації дефекту;*
- *тест-кейси для повторного тестування можуть бути виявлені після першого раунду тестування;*
- *тест-кейси для повторного тестування неможливо знайти автоматизовані;*
- *вимагає додаткового часу для проходження вже пройдених раніше тест-кейсів.*

Регресійне тестування (regression testing)

Переваги:

- *підтверджує відсутність багів після додавання фічі або редагування коду;*
- *може бути виконано з використанням інструментів автоматизації;*
- *допомагає покращити якість продукту.*
- *Недоліки:*
- *Цей вид тестування може бути дуже трудомістким.*

Структурне тестування

Структурне тестування спрямовано тестування структури системи чи компонента. Цей вид тестування, як правило, відносять до тестування «білого» та «сірого» ящиків, тому що ми перевіряємо, що відбувається всередині системи або програми.

Методи структурного тестування:

Строкове покриття (Statement Coverage) – перевірка застосування всіх операторів у програмі використання (хоча б один раз);

Покриття шляху (Path Coverage) – призначений задоволення критеріїв охоплення кожного логічного шляху через програму;

Покриття рішення (Branch Coverage) – перевіряє, чи кожна умова розгалуження для програми є справжніми або помилковими значеннями;

Покриття умови (Condition Coverage) – схоже на Branch Coverage, основна відмінність полягає у перевірці стану покриття для умовних та не умовних гілок.

Переваги:

- можливість виявити та видалити «зайвий» код;
- Можливість виявлення потенційних помилок на ранній стадії;
- Забезпечує більш ретельне тестування ПЗ;

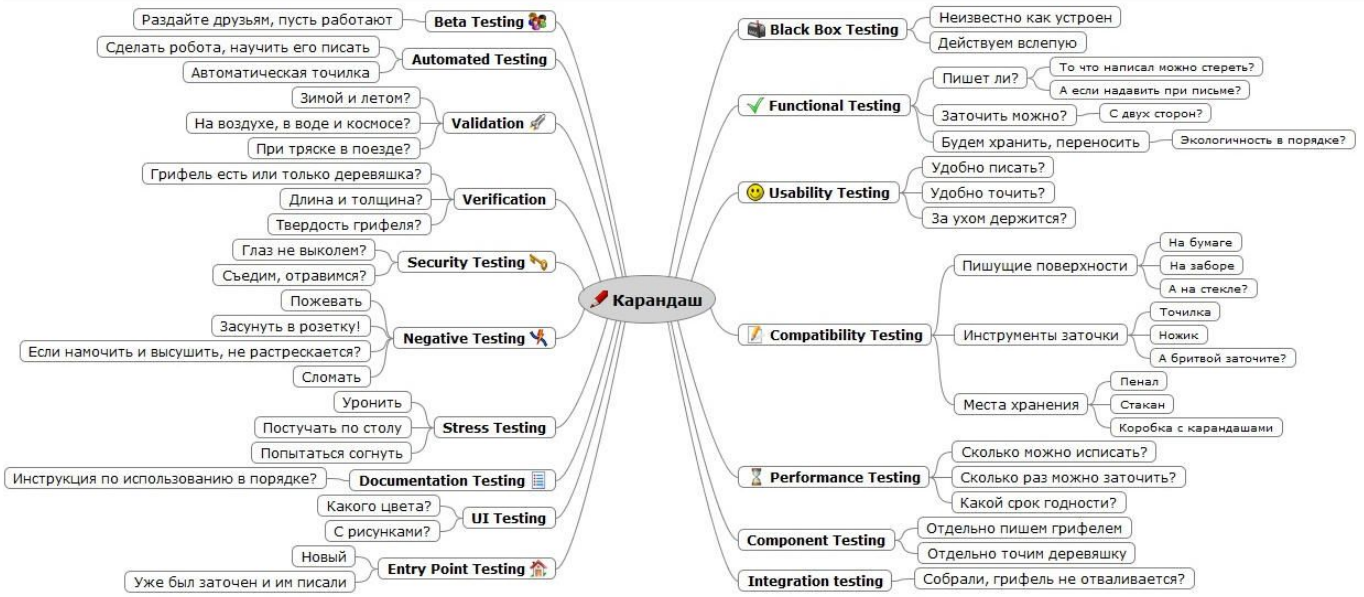
Недоліки:

- вимагає знання коду та інструментів тестування.

Домашнє завдання

Вибрати будь-який предмет (чашка, телефон, машина тощо) та за аналогією з картинкою нижче описати різні види його тестування. Оформити можна у вигляді схеми (як на зображенні) або у вигляді файлу exel, завантажити його на платформу.

Рекомендація: не прив'язуватися до списку видів тестування із зображення, а подумати які види тестування будуть актуальні для вашого обраного об'єкта.



Корисні посилання

1. https://training.qatestlab.com/blog/technical-articles/alpha-beta-testing/
2. https://geteasyqa.com/ru/qa/software-testing-types/