


КОНСПЕКТ УКРАЇНСЬКОЮ JUNIOR ТЕСТУВАЛЬНИК ПЗ

Підготувала Popeliuha

 <https://www.youtube.com/c/PopeliuhaQA>

 <https://t.me/popeliuha>

 <https://instagram.com/popeliuhaqa>

 <https://www.tiktok.com/@popeliuha>

Даний конспект - це збірка з багатьох джерел з інтернету. В більшості це переклад російської версії шпаргалки авторства Наталії Матвєєвої, але я додала сюди і свої знання :) Завжди пам'ятайте, що скільки людей - стільки теорій. Основні документи, з яких офіційно береться інформація - силабуси ISTQB та стандарт IEEE 29119-3.

Тестування Програмного Забезпечення (Software testing).

- це перевірка якості програмного забезпечення;
- перевірка відповідності актуального результату до очікуваного результату;
- пошук дефектів;
- переконання в якості продукту;
- запобігання виникненню дефектів;
- перевірка робочих продуктів (документів створених в процесі тестування);
- перевірка на дотримання вимог;
- зменшення ризиків;
- перевірка, що програма відповідає юридичним, договірним і нормативним вимогам та стандартам.

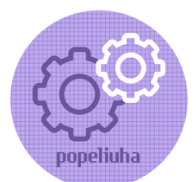
Інформація взята з Силабусу ISTQB Foundation 2018 р.

Навіщо тестувати Програмне Забезпечення?

Помилки можуть виникати з багатьох причин. Наприклад,

- не достатньо часу на розробку;
- люди можуть помилятися - людський фактор;
- недосвідчені або недостатньо кваліфіковані учасники проєкту;
- непорозуміння між учасниками команд;
- незрозумілі вимоги;
- складність коду, архітектури;
- нерозуміння міжсистемних інтерфейсів;
- нові і незнайомі технології.

Інформація взята з Силабусу ISTQB Foundation 2018 р.



ТЕСТУВАЛЬНИК

Тестувальник - це спеціаліст, який перевіряє якість програмного забезпечення та його відповідність вимогам.

Тестувальник шукає дефекти в програмах та повідомляє про них.

QUALITY ASSURANCE, QUALITY CONTROL, TESTING

Quality Assurance engineer - інженер із забезпечення якості. Займається тестуванням програми у всіх фазах життєвого циклу програмного забезпечення. Найбільше уваги в роботі приділяє процесам, але також займається тестуванням продукту.

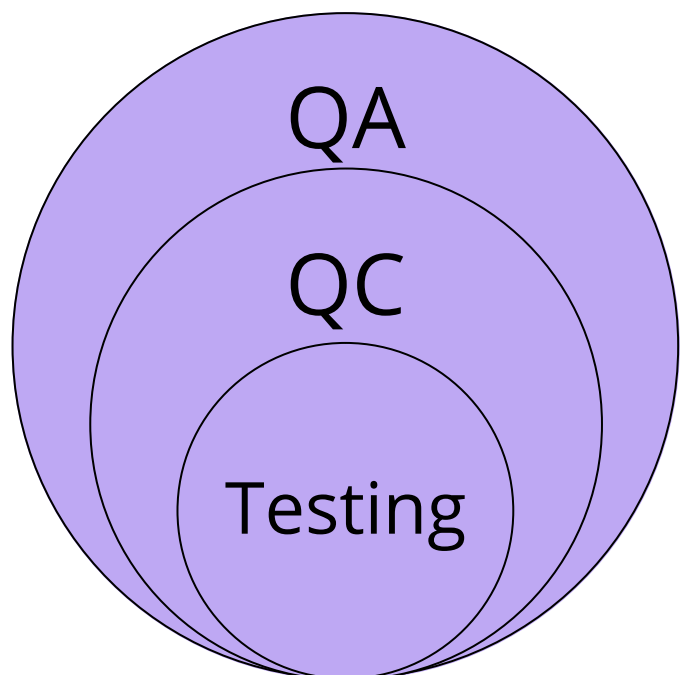
Quality Control engineer - інженер з контролю якості. Займається тестуванням програми у фазах розробки та тестування. На роботі займається продуктом (додатком, сайтом).

Тестувальник - працює у фазі тестування, шукає дефекти в програмі.

Професія QC поглинає обов'язки Тестувальника, а QA поглинає обов'язки QC.

В реальному житті ці поняття постійно плутають.

У назві вакансії може бути вказано QA, коли шукають тестувальника. Не варто цього боятись, обов'язкам QA інженера легко навчитись.



Принципи тестування

1, Testing shows the presence of defects, not their absence

Тестування демонструє наявність дефектів, але не гарантує їх відсутності. Тестування тільки зменшує вірогідність наявності дефектів, що знаходяться в програмі.

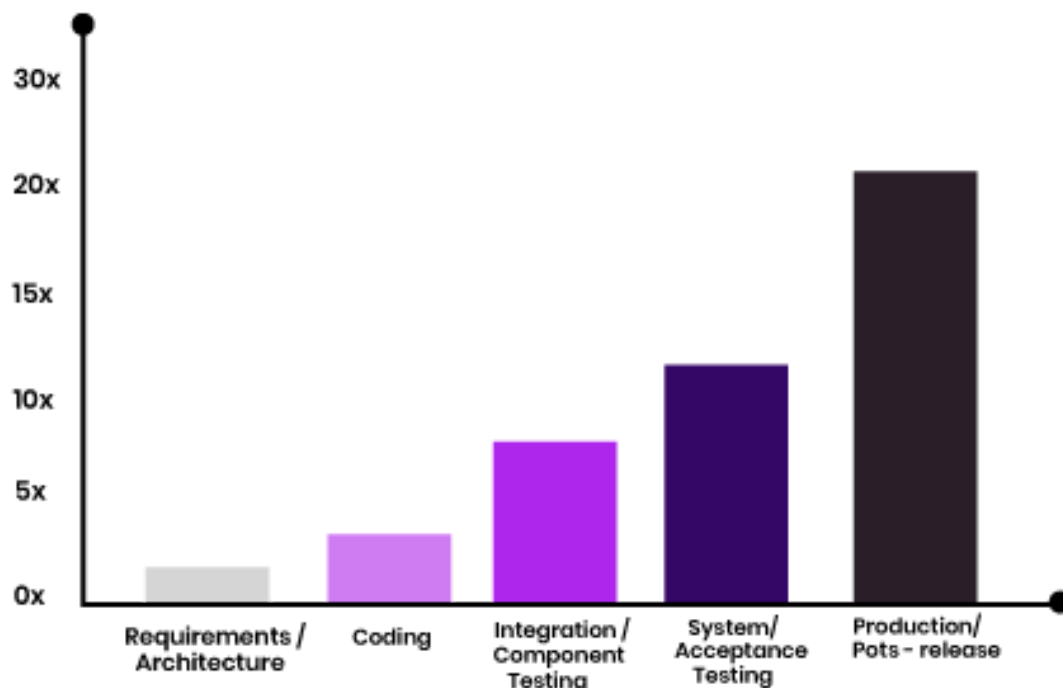
2. Exhaustive testing is not possible

Вичерпне тестування неможливе. Повне тестування з використанням всіх вхідних комбінацій даних, результатів і передумов фізично неможливе. Тому при створенні тестів треба враховувати ризики і використовувати техніки тестування.

3. Early testing saves time and money

Раннє тестування. Тестування слід починати на ранніх стадіях життєвого циклу розробки програмного забезпечення, щоб знайти дефекти якомога раніше. Чим пізніше знайдено баг, тим дорожче він коштує.

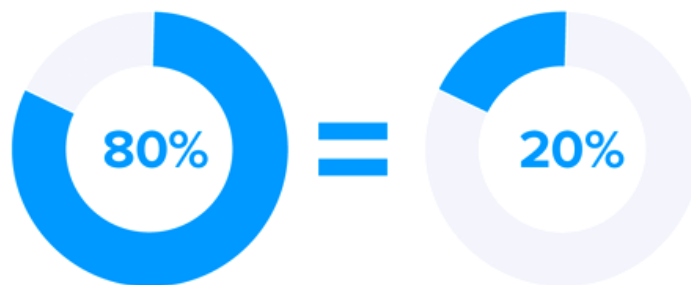
Вартість виправлення дефектів залежно від часу виявлення:



Принципи тестування

4. Defects cluster together

Накопичення дефектів. Більшість дефектів знаходяться в обмеженій кількості модулів. Кажуть, що цей принцип працює як принцип Парето. Принцип Парето — емпіричне правило, яке стверджує, що для багатьох явищ 80 відсотків наслідків спричинені 20 відсотками причин. Мається на увазі, що 80% дефектів розташовані в 20% модулів.



5. Beware of the pesticide paradox

Парадокс пестициду. Якщо повторювати ті самі тести знову і знову, то в певний момент цей набір тестів перестане виявляти нові дефекти. Тому тест кейси постійно треба оновлювати і додавати. Принцип має таку назву через аналог до роботи пестицидів: якщо постійно використовувати той самий пестицид, то шкідники отримають до нього імунітет.

6. Testing is context dependent

Тестування залежить від контексту. Наприклад, програмне забезпечення, в якому критично важлива безпека, тестується інакше, ніж сайт-візитка.

7. Absence of errors is a fallacy

Омана про відсутність помилок. Відсутність знайдених дефектів під час тестування не завжди означає готовність продукту до релізу. Система повинна бути зручна для користувачів і має задовільняти їх очікування та потреби.

Цикл розробки ПЗ (Software development lifecycle, SDLC).

Кожна програма має свою історію створення і ця "історія" завжди складається з однакових кроків або фаз. Ці фази добре зображено на діаграмі:

На фазі Планування (1) визначають "що ми хочемо зробити", скільки ресурсів (людей, ПК, ліцензій) на це потрібно.

На фазі Аналізу (2) аналізують вимоги, визначають "які проблеми потребують рішення".

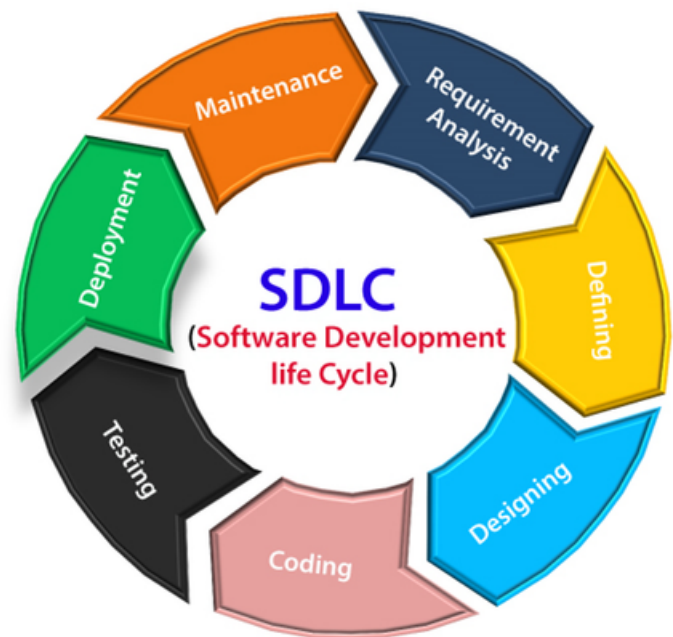
На фазі Дизайну (3) проектується майбутня програма: розробники створюють моделі і блок-схеми майбутнього коду, а тестувальники проектують майбутні Тест Кейси.

На фазі Імплементації або Розробки (4) розробники пишуть код і створюють програму.

На фазі Тестування (5) тестувальники перевіряють програму на якість, наявність дефектів та відповідність вимогам.

Фаза Впровадження, або Деплоймент зазвичай іде наступною(6) (але на картинці вона об'єднана з Тестуванням). В цій фазі готову версію програми "віддають" клієнту в користування або "деплоять" в мережу.

На фазі Підтримки (7) виправляють існуючі баги та створюють нові версії програми.



Цикл тестування ПЗ (Software testing lifecycle, STLC).

Окрім SDLC є ще цикл Тестування програмного забезпечення. В ньому описуються всі дії тестувальника відповідно до процесу створення ПЗ.

Життєвий Цикл Тестування ПЗ - це порядок дій, завдяки яким гарантується якість ПЗ і забезпечується його відповідність вимогам.

Відповідно до ISTQB, цикл Тестування складається з наступних фаз:

- 1.Планування
- 2.Моніторинг та контроль
- 3.Аналіз та дизайн
- 4.Реалізація
- 5.Виконання тестування
- 6.Завершення тестування

Для кращого розуміння розберемо, що роблять тестувальники на кожній фазі.

Під час Планування (1) тестувальники: 

- Визначають масштаб роботи, ризики та цілі тестування
- Визначають загальний підхід до тестування
- Планують тестові заходи, виділяють ресурси для роботи
- Визначають кількість документації та шаблони для неї
- Обирають метрики для Моніторингу та Контролю
- Ухвалюють рішення про автоматизацію
- Визначають entry та exit критерії

Entry criteria (критерії входу в тестування) визначають умови, які мають бути виконані до початку робіт.

Exit criteria (критерії виходу з тестування) визначають, які умови мають бути виконано, щоб завершити рівень тестування або набір тестів.

Цикл тестування ПЗ (Software testing lifecycle, STLC).

Моніторинг та Контроль (2):

- Моніторинг - це процес вимірювання прогресу на проєкті. Прогрес вимірюється в метриках - формулах, які обчислюють успіх, певний відсоток покриття тощо.
- Контроль тестування включає в себе вживання дій, щоб слідувати тест плану.
- Також до контролю тестування відноситься ініціація коректив, плану Б (на випадок якщо головний план не виконується)

Фаза Аналізу (3) містить такі активності:

- Аналіз тестового базису (документів, з яких створюють Тест Кейси)
- Визначення набору функцій, які підлягають тестуванню, і які неможливо протестувати
- Визначення умов тестування, ґрунтуючись на аналізі базису тестування
- Врахування функціональних, нефункціональних та структурних характеристик, інших технічних факторів, рівнів та ризиків

Під час фази Дизайну або Проєктування (3) необхідно:

- Проєктування і пріоритизація тест кейсів
- Визначення потрібних тестових даних для підтримки тестових умов та тест кейсів
- Проєктування тестового середовища та визначення потрібних програм
- Створення матриці відслідковування (traceability matrix) між тест базисом і тест кейсами

Test basis - сукупність знань, яка використовується як основа для аналізу та розробки тестів. Простими словами, це всі документи, з яких можна створити тести (вимоги, специфікації, інструкції).

Цикл тестування ПЗ (Software testing lifecycle, STLC).

Реалізація тестування (4) містить такі активності:



- Розробка та пріоритизація тестових процедур
- Створення наборів тестів - тест сьютів
- Автоматизація сценаріїв, якщо вона необхідна
- Організація розкладу виконання наборів тестів
- Налаштування тестового середовища
- Підготовка тестових даних та їх завантаження в тестове середовище
- Перевірка і оновлення відслідковуваності між базисом тестування, тестовими умовами, сценаріями та сьютами

Виконання (5) тестування - це:



- Запис ID та версій об'єктів тестування і програм для тестування
- Виконання тестів мануально або за допомогою програм
- Порівняння актуальних та очікуваних результатів
- Аналіз проблем з метою вирішити х причину
- Звітування дефектів, базуючись на впавших тестах
- Логування результатів виконання тестів

В фазу Закриття тестування (6) входить:



- Перевірка закриття всіх звітів про дефекти, списку завдань, які залишилися не реалізованими в кінці тестування.
- Створення звіту з тестування для передачі зацікавленим особам
- Завершення та архівування тестового середовища, тестових даних для подальшого використання
- Аналіз отриманих уроків для визначення змін, необхідних для майбутніх ітерацій, релізів та проектів
- Використання зібраної інформації для покращення процесу тестування

Моделі та Методології розробки Програмного Забезпечення

Методологія розробки ПЗ - це набір методів та критеріїв оцінки, які використовуються для постановки задач, планування, контролю та для досягнення поставленої цілі.

Сам процес розробки описується **моделлю**, що визначає послідовність найбільш загальних етапів і отриманих результатів.

Модель життєвого циклу ПЗ - структура, що визначає послідовність виконання і взаємодії процесів, дій та задач протягом життєвого циклу.

Модель обирають, орієнтуючись на:

- Напрямок проекту
- Масштаб проекту
- Бюджет
- Терміни реалізації кінцевого продукту

Водоспадна модель або Waterfall

Waterfall - це модель, в якій всі стадії життєвого циклу розробки ПЗ повинні проводитися в суворому порядку, без повторень, кожна стадія повинна завершитися перед початком наступної. Найчастіше використовується в медичних та військових проектах.

Переваги: вартість і дедлайни визначені заздалегідь.

Недоліки: неможливо повернутись на попередню фазу або повторити фазу; неможливо змінити вимоги під час розробки.

Планування

Аналіз вимог

Дизайн

Розробка

Тестування

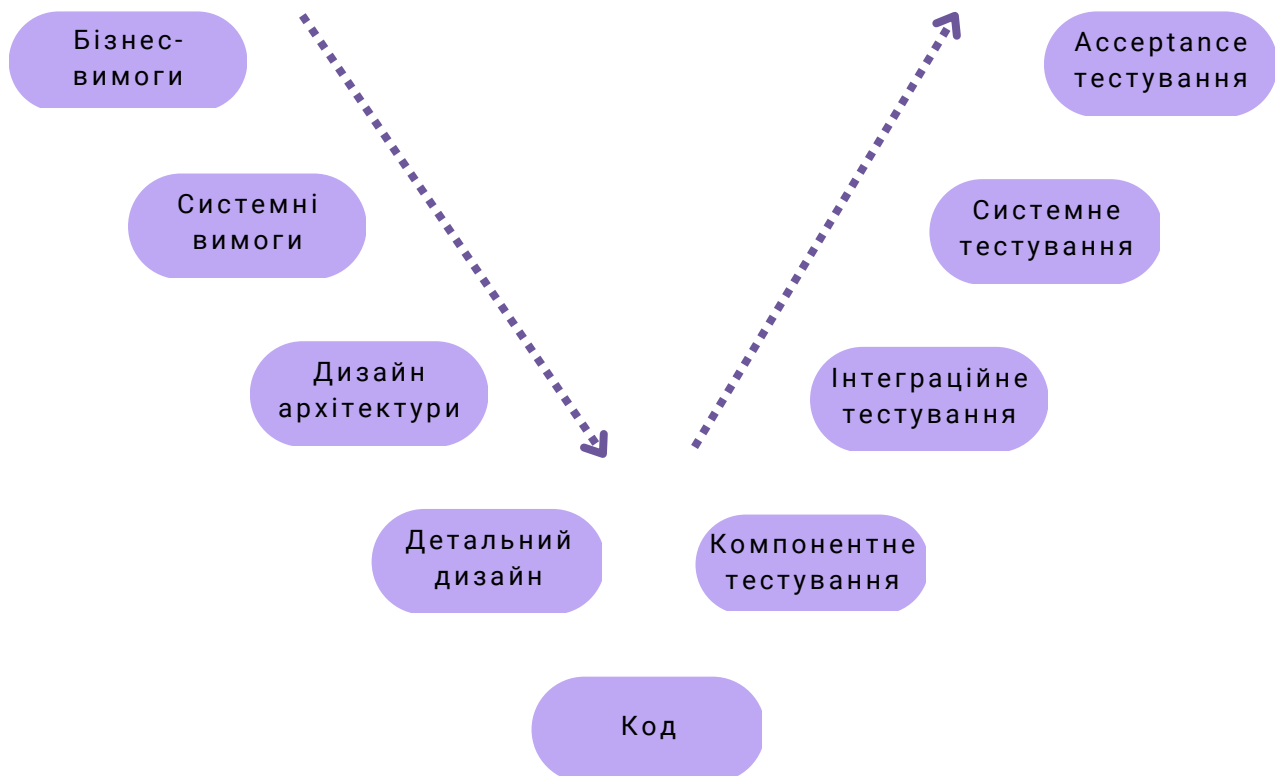
Реліз

Підтримка



V - модель

V-модель відома тим, що кожному кроку команди розробки відповідає певний крок або дія тестувальників. Якщо поглянути на схему V-моделі, то видно, що дії в лівій гілці виконують бізнес аналітики та розробники, а дії справа - команда тестування.

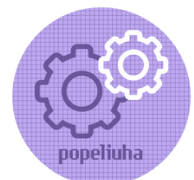


Переваги моделі:

- Тестування починається на ранніх стадіях розробки ПЗ
- Кількість помилок в архітектурі зменшується
- З кожною стадією розробки проводиться стадія тестування

Недоліки моделі:

- Якщо під час розробки програмного забезпечення було допущено помилку, то повернутись і виправити її буде дорого коштувати, як в "водоспаді"



Ітераційна / Інкрементальна модель

Ітеративність - повторення операцій з метою переробки результатів попереднього етапу.

Інкрементальність - додавання результатів до попереднього етапу.

При цьому підході проект проходить цикл, що повторюється (ітерації):

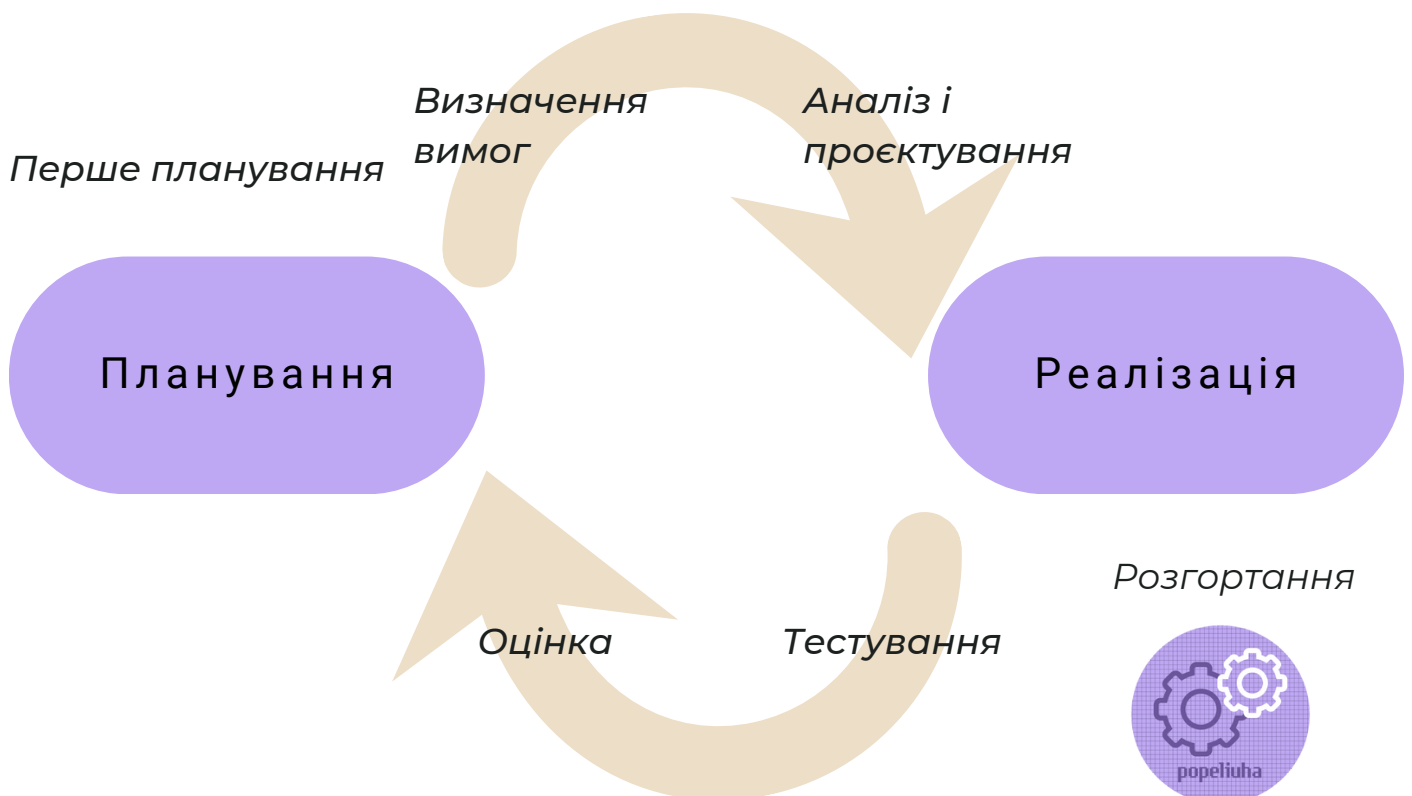
Планування - Реалізація - Перевірка - Коректування

Переваги:

- Гнучкість в прийнятті нових вимог та змін
- Можливість адаптації процесу на основі уроків, засвоєних з попередніх ітерацій
- Коротші терміни виводу продукту на ринок

Недоліки:

- Вартість продукту невідома
- Можуть виникати проблеми з архітектурою системи, оскільки вимоги для всього життєвого циклу програми не збираються.



Спіральна модель

Спіральна модель - це модель процесу розробки ПЗ з врахуванням ризиків. Це комбінація моделі водоспаду і ітеративної моделі.

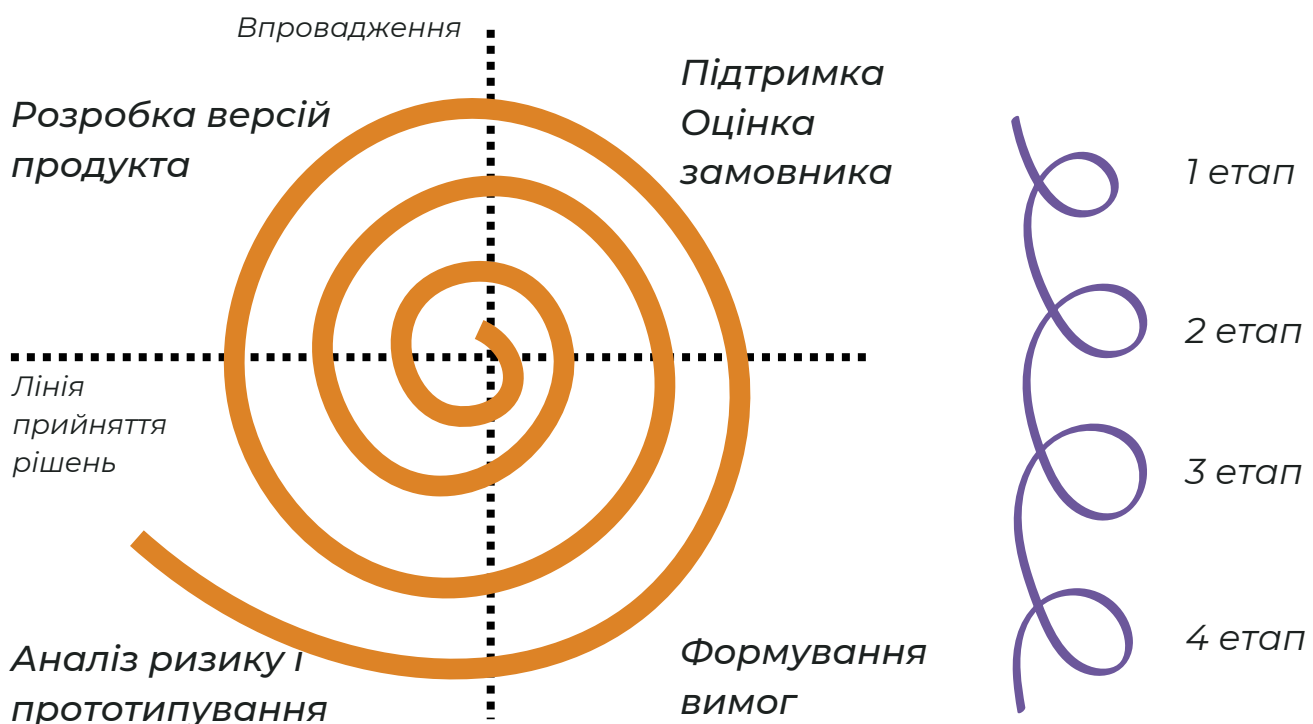
Кожна фаза (ітерація) спіральної моделі в розробці програмного забезпечення починається із визначення цілі проектування і закінчується тим, що клієнт переглядає прогрес.

Переваги:

- Аналіз ризиків і управління ризиками на кожному етапі
- Підходить для великих проектів
- Можливість зміни в вимогах на пізніх етапах
- Замовник може спостерігати за розвитком продукту на ранній стадії розробки

Недоліки:

- Не підходить для невеликих проектів, бо вона дорога
- Успішне завершення проекту залежить від аналізу ризиків
- Кількість етапів невідома на початку проекту



Гнучкий принцип розробки Agile

Принцип Agile - це набір практик для оперативної реакції на зміни в ході робочого процесу. Такі підходи допомагають швидко реагувати на фідбек клієнтів і замовників, постійно покращуючи продукт. Принципи маніфесту:

<https://agilemanifesto.org/iso/uk/principles.html>

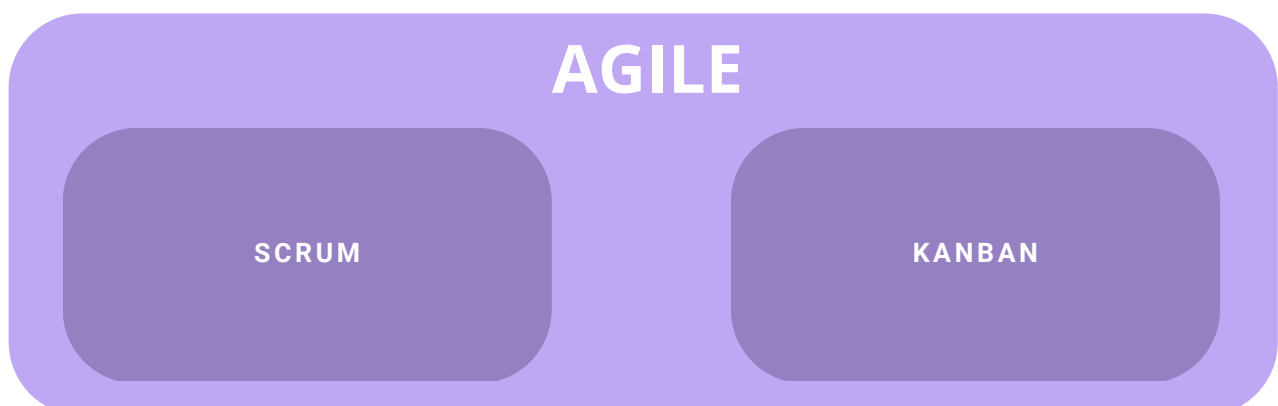
Цінності Agile

1. Люди та співпраця важливіші за процеси та інструменти
2. Працюючий продукт важливіший за вичерпну документацію
3. Співпраця із замовником важливіша за обговорення умов контракту
4. Готовність до змін важливіша за дотримання плану

Суть Agile маніфесту

- Вся робота над проєктом розділяється на короткі цикли (ітерації) і ведеться поетапно;
- В кінці кожної ітерації замовник отримує готовий мінімально робочий продукт чи його частину, яку вже можна використовувати;
- Протягом всього робочого процесу команда співпрацює з замовником;
- Будь-які зміни в проєкті вітаються і швидко інтегруються в роботу.

Є 2 основних фреймворка, що засновані на базових принципах Agile: Scrum і Kanban.



SCRUM

Підхід передбачає взаємодію команди розробки, Product Owner`а та Scrum-майстра на кожному етапі розробки ПЗ.

Суть даного фреймворку:

- Робота ділиться на спринти тривалістю 1-4 тижні
- Перед початком спринта команда сама формує список задач на ітерацію
- Під час кожного спринта створюється продукт чи послуга, які можна продемонструвати клієнту
- Мітинги, які допомагають в процесі роботи: Щоденний стендап, Планування, Ретроспектива та Sprint Review.
- Обов'язкові ролі: Scrum-master (допомагає вирішити блокери, слідкує за дотриманням скраму) та Product Owner (виставляє пріоритети і цілі спринта)

KANBAN

Канбан – це спосіб правильного налаштування процесу з метою максимально ефективно використати можливості кожного співробітника. Підхід дозволяє оптимізувати роботу команди через розділення об'ємних етапів на окремі операції.

Система Канбан заснована на принципах:

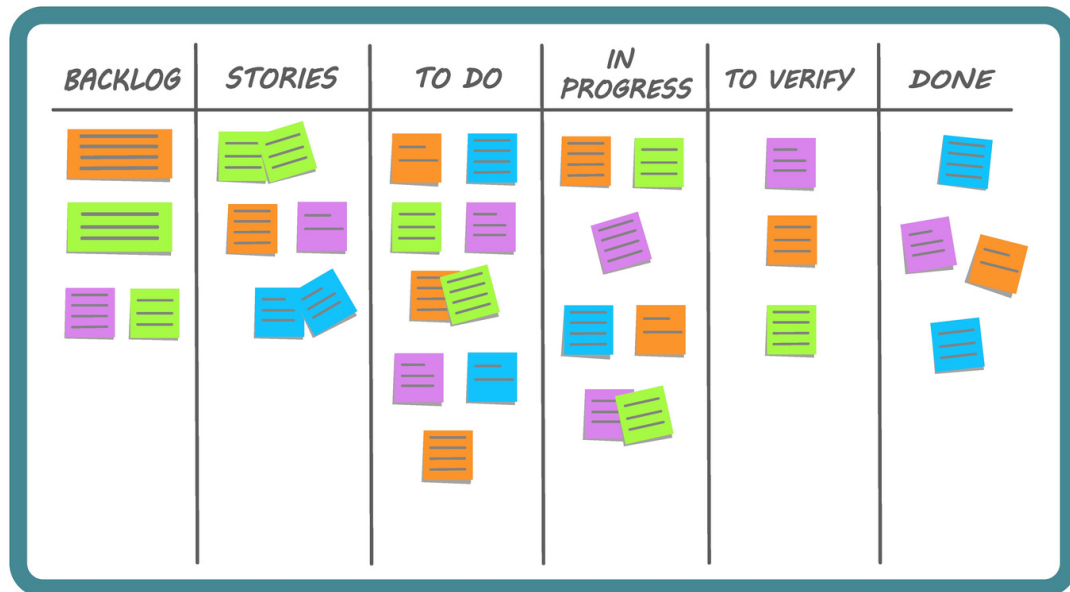
- Візуалізація. Основа Канбан – візуальна дошка, на якій представлені етапи виконання поточної задачі: «заплановано» / «виконується» / «зроблено».
- Розподіл завдань. Чітке розуміння цілей спрощує процес.
- Фокусування на роботі. Якщо процес затягується, потрібно підключити додаткових співробітників і перерозподілити ресурси.

Основні відмінності:

- В Канбані може не бути спринтів. Завдання і пріоритети можна змінювати під час роботи, тоді як в Scrum - тільки до початку наступного спринта.
- В Канбані немає таких ролей, як в Scrum. Може бути тільки команда і Тех Лід.
- В Scrum мітинги є обов'язковими, а в Канбані - ні.

Scrum-дошка та Backlog

Scrum дошка - це місце, де IT-команди тримають завдання. Дошка складається з різних колонок, наприклад, "Зробити", "В процесі", "Потребує перевірки", "Зроблено".



Scrum дошка має бути видимою будь-кому в команді і поза нею (навіть замовникам). Під час спринта учасники можуть рухати свої завдання у відповідні колонки. Завдання беруться тільки на 1 спринт.

- To Do - потрібно зробити
- In progress - в процесі
- Ready for review (або To verify)- чекають на перевірку
- Done - готово

Беклог продукту (product backlog) – це упорядкований набір завдань, які зацікавлені люди хочуть отримати від продукту. Цей список містить короткі описи всіх бажаних можливостей продукту.

Беклог спринта – набір елементів із беклога продукту для виконання в поточному спринті.

Вимоги

Вимоги — це опис того, що робить або має робити конкретне програмне забезпечення. Вони використовуються під час розробки, щоб повідомити, як функціонує програмне забезпечення або як воно призначене для роботи.

РІВНІ ВИМОГ

- **Бізнес-вимоги**
 - Навіщо створюється продукт?
 - Яка користь з продукту?
 - Як отримати прибуток з продукту?
- **Користувацькі вимоги**
 - Що користувач може робити з допомогою продукту?
- **Продуктові вимоги**
 - Функціональні: Що система повинна робити?
 - Нефункціональні: Як система має це робити?
 - Продуктивність
 - Документація
 - Системне середовище

User story - це "неформальні вимоги", які допомагають зрозуміти потреби користувача і складаються з 3х частин:

Як *(роль користувача)*, я *(хочу щось отримати)*, для того
щоб *(причина)*
(Як Адміністратор, я хочу додавати людей в бан, щоб
уникнути образ та спаму.)

Acceptance criteria - це набір прийнятих умов або ділових правил, яким повинна відповідати функціональність, щоб її прийняли Власник продукту / Зацікавлені сторони.

Given ... when ... then ... - мова Gherkin в програмі Cucumber

Given Обрані товари When користувач натискає "Додати в Обране" Then товар додається до обраного.

Тестування вимог

Інколи тестувальникам доводиться перевіряти не тільки програму, а і вимоги. Ціль тестування – усунути можливі помилки на початкових етапах проектування системи, знизити підсумкову вартість та покращити якість продукту.

Характеристики хороших вимог:

- Повнота. Вимоги повинні в повній мірі описати функцію, яку необхідно реалізувати.
- Необхідність. Вимоги повинні описувати дійсно необхідний функціонал.
- Реальність виконання. Вимоги повинні описувати, наскільки можливо реалізувати функціонал.
- Відсутність протиріч. Різні документи або тексти в одному документі не повинні конфліктувати один з одним.
- Однозначність. Потрібно уникати розбіжностей в вимогах і не використовувати “плаваючі” слова.
- Грамотність. Потрібно уникати граматичних і логічних помилок у вимогах.
- Здійснюваність. Наскільки вимоги можна реалізувати.
- Можливість перевірити всі прописані вимоги після реалізації.
- Модифікованість. Ця властивість характеризує простоту внесення змін до окремих вимог і набір вимог.
- Атомарність. Вимога описує лише одну ситуацію.
- Простежуваність.
- Актуальність.

Чек-ліст хороших вимог:

- 1.Перевірити наявність слів “і так далі”, “та\або”, “може”, “і не тільки”, “коли потрібно”, а також “я, ти, ми, ви, цей, ці”
2. Перевірити на двозначність
- 3.Перевірити наявність слів, що неможливо перевірити: “швидкий, зручний, простий, якісний, гнучкий” і подібних
- 4.Перевірити на вимірюваність
- 5.Перевірити на необхідність (без вимоги можна обійтись?)
- 6.Перевірити опис реалізації (вимоги описують що повинно працювати, а не як це має бути реалізовано)

Тестова документація

Тестова документація – це набір документів, що створюється протягом усього циклу тестування. Документація допомагає команді однозначно трактувати кроки, терміни тестування, результати, звертатися до цієї інформації у спірних моментах.

Види тестової документації:

- **Check list:** Список, що містить ряд необхідних перевірок.
- **Test case:** Набір кроків для виконання та очікуваних результатів, розроблений для детальної перевірки функціоналу.
- **Test suite:** Набір test cases для компонента чи системи.
- **Test summary report:** Документ, що звітує висновок по результатам тестування.
- **Defect report:** Документ, що містить звіт про недолік в програмі, який може привести її до неможливості виконати потрібну функцію.
- **Test plan:** Документ, що описує цілі, підходи, ресурси та графік запланованих тестових активностей. Він визначає об'єкти тестування, завдання, відповідальних за завдання, тестове середовище, визначає критерії входу та виходу а також будь-які ризики, що вимагають планування.
- **Test strategy:** Високорівневий опис рівнів тестування, що повинні бути виконані, і тестування, що входить в ці рівні, для організації програми із **одного чи декількох проектів**.
- **Test policy:** Документ високого рівня, що описує принципи, підхід та основні цілі **організації** в тестуванні.

Чек-ліст

- Чек-ліст - таблицка, яка складається з назви перевірки, статусу та тестових даних або коментаря. Назви кроків є лаконічними, зрозумілими і завжди починаються з дієслова. Наприклад, "Додати користувача в друзі"
 - Статуси існують наступні: Passed, Failed, Blocked, In progress, Skipped, Not run
- В тестові дані вводимо все, що користувач обирає або вводить з клавіатури (ім'я, вік, адреси, паролі, дати, назви товару тощо)

Тест Кейс

Тест кейс - це документ, що являється інструкцією тестування і включає в себе кроки для виконання тестової ситуації і очікуваний результат.

Тест кейс - це більш детальна і формальна документація, ніж **чек-ліст**, і використовується частіше.

Набір тест-кейсів називається **Test Suite**. Зазвичай тест кейси групують по темі або сторінці, яку вони тестують.

Тест кейс складається з:

- **ID** - унікальний ідентифікатор тест-кейсу. Його зручно використовувати для легкого пошуку тест-кейсу.
- **Summary (Назва)** - короткий опис суті перевірки. Починається з дієслова.
- **Pre-conditions (Передумови)** - опис дій, які необхідно виконати перед основними кроками. Наприклад, відкрити сайт, мати зареєстрований аккаунт.
- **Steps (Кроки)** - дії, які необхідно виконати для перевірки.
- **Post-conditions (опціонально)** - дії, які необхідно виконати, щоб повернути систему в попередній стан. Наприклад, видалити користувача.
- **Test data** - всі дані, які користувач вводить або обирає. Наприклад, Ім'я, номер телефону тощо.
- **Screenshots** - кожен крок має включати скріншот, щоб було зрозуміло, які дії проводити.
- **Expected result** - те, що ми очікуємо побачити в результаті перевірки.
- **Requirement link**. Посилання на вимогу або ТЗ, на основі якої було складено тест-кейс.
- **Author** - тестувальник, який написав тест-кейс.
- **Priority** – наскільки важливим є цей тест-кейс, в яку чергу його варто виконувати.
- **Product version** – опис програмного забезпечення, на якому можна виконати тест-кейс.

Тест-репорт (Test report) – звіт про виконання тест-кейсів, у ньому зазвичай зображена загальна статистика, кількість виконаних тест-кейсів та кількість знайдених помилок, висновки щодо релізу.

Тест план і стратегія

Тест план (Test Plan) – документ, що описує весь обсяг робіт із тестування: опис об'єкта тестування, стратегії, критеріїв початку та закінчення тестування, необхідне обладнання та знання, оцінки ризиків з варіантами їх вирішення.

Тест стратегія (Test Strategy) – це документ високого рівня, який містить вказівки та принципи, пов'язані з проведенням процесу тестування.

| Тест План | Тест Стратегія |
|---|--|
| Створює Test Manager або Test Lead | Створює Project Manager |
| В деталях описує об'єм тестування і дії, необхідні для проведення тестування | Високорівневий (не детальний) документ з інструкціями про те, як буде проходити тестування |
| Описує всі активності тестування: які використовувати техніки, графік, ресурси тощо | Поверхньо описує техніки, що будуть використовуватись, систему для тестування тощо |
| Тест плани можуть бути змінені і оновлені | Тест стратегію зазвичай не змінюють |
| Тут багато деталей та уточнень | Тут загальні підходи та методології |

Немає чіткого шаблону, за яким необхідно писати тест-план. Головне лише те, що він має описати весь обсяг робіт із тестування та бути зрозумілим та читабельним.

На практиці тестова документація ще називається **тестовими артефактами** чи **артефактами тестування**.

Артефакт – це щось створене чи використане набором тестів.

Наприклад,

Файл логу є артефактом.

Якщо ваш тест завантажує зображення, то це артефакти.

Артефактом може бути рядок, доданий до бази даних.

Дефекти

Баг (bug, дефект) – відхилення фактичного результату від очікуваного. Знайдені баги оформлюються в баг-репорти.

Очікуваний результат (Expected result) – опис того, як саме має працювати система відповідно до документації.

Фактичний результат (Actual result) – це результат, який отримує тестувальник під час тестування. Тобто те, як система працює насправді.

Якщо **Баг** – це недогляд, то **Фіча** – спеціально передбачена можливість. Популярна фраза "**Не баг, а фіча**" полягає в тому, що **баг** - випадково припущена помилка, що заважає нормально користуватися програмою, а **фіча** - спеціально зроблена функція, незвичайна для інших програм такого типу.

Баг-репорт (Bug Report – звіт про помилку) – це технічний документ, тому він створюється за певними правилами.

Баг-репорт складається з:

- 1.ID - унікальний ідентифікатор
- 2.Назва (відповідає на питання Що? Де? Коли?)
- 3.Передумови, кроки для виконання, післяумови
- 4.Актуальний та очікуваний результати
- 5.Пріоритет та ступінь впливу (Priority, Severity)
- 6.Тип дефекту
- 7.Вкладення (скріншоти, логи, відео)
- 8.Система, на якій знайдено баг
- 9.На кого назначений баг-репорт і його статус

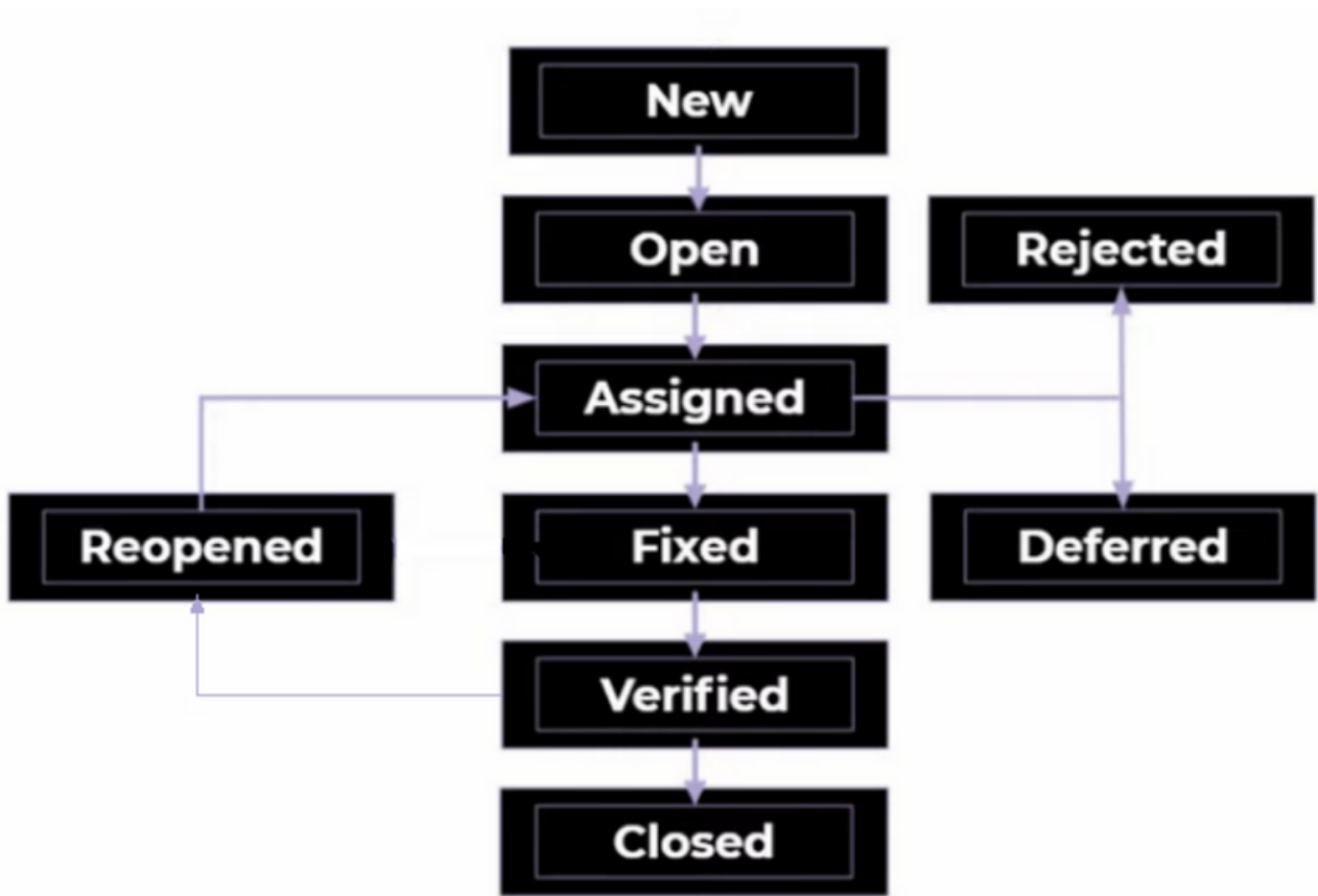
Severity

- Наскільки баг “зламав” функціональність
- Виставляє тестувальник
- Блокер: Не працює реєстрація. Не завантажується основна сторінка
- Косметичний: Не перекладено текст.

Priority

- Як швидко треба виправити баг
- Впливає на репутацію клієнта
- Виставляє менеджер / продакт
- Високий: помилка в лого компанії
- Низький: граматична помилка.

Життєвий цикл дефектів



Життєвий цикл Баг-репортів:

- New - вперше знайдено баг і внесено до системи.
- Open - взято в роботу.
- Assigned - назначено розробника, який буде виправляти баг.
- Rejected (Not a bug) - розробник відхилив баг або не зміг відтворити. Ми маємо перевірити, чи баг відтворюється, і якщо так, то показати це розробнику.
- Deferred - перенесено на наступні спринти через брак часу і низький пріоритет.
- Fixed - розробник виправив баг і чекає на перевірку QA.
- Verified - тестувальник перевірів баг.
- Reopened - якщо баг не виправлено і він відтворюється, його перевідкривають.
- Closed - якщо баг виправлено, його закривають.

Статичне і динамічне тестування

Статичне тестування – під час статичного тестування код не виконується (програма не запускається).

Перевіряються:

- Документи (вимоги, тест-кейси, описи архітектури, схеми баз даних тощо).
- Графічні зразки (дизайн).
- Код програми (виконується програмістами в рамках code review).
- Параметри (налаштування) середовища виконання програми.

Статичне тестування починається на ранніх етапах життєвого циклу і є, відповідно, частиною процесу **верифікації**.

Динамічне тестування – тестування із запуском коду виконання, як всього додатку цілком (системне тестування), так і кількох взаємозалежних частин (інтеграційне тестування), окремих частин (модульне тестування) і навіть окремі ділянки коду.

Динамічне тестування включає тестування ПЗ в режимі реального часу шляхом надання вхідних даних і вивчення результату поведінки ПЗ. Перевірка здійснюється за допомогою попередньо підготовленого набору тестів. Є частиною процесу **валідації** програмного забезпечення.

Верифікація і Валідація

Верифікація – процес оцінки системи або її компонентів з метою визначення, чи задовольняють результати поточного етапу розробки сформованим на початку етапу вимогам.

Валідація – перевірка відповідності ПЗ до вимог користувача. Програма може на 100% відповідати специфікації, але при цьому виконувати зовсім не те, що хотів користувач/замовник.

Black box, White box, Gray box

White Box testing: у тестувальника є доступ до внутрішньої структури та коду програми, а також є достатньо знань для розуміння побаченого.

Black Box testing: у тестувальника або немає доступу до внутрішньої структури та коду додатка, або недостатньо знань для їх розуміння, або він свідомо не звертається до них у процесі тестування. В рамках тестування за методом чорної скриньки основною інформацією до створення тест-кейсів є документація.

Gray Box testing: комбінація методів білої скриньки та чорної скриньки, що полягає в тому, що до частини коду та архітектури у тестувальника доступ є, а до частини — ні. Наприклад, є доступ до API та бази даних, але нема доступу до коду.

Мануальне та Автоматизоване

Мануальне (ручне) тестування – це тестування без допомоги будь-яких програм, що автоматизують роботу. Будується на методах тестування – сюди належать і техніки тест-дизайну, і техніки, що базуються на досвіді.

Автоматизоване тестування – метод тестування програмного забезпечення з використанням спеціальних програмних засобів.

Позитивне і негативне

Позитивне тестування – це тестування, в якому вводимо правильні дані, очікуємо хороший результат

Негативне тестування – це тестування, в якому вводимо невірні дані, очікуємо повідомлення про помилку. Всі негативні випадки протестувати неможливо, але пара ключових тестів - must have.

Спочатку виконуємо позитивне тестування, а потім негативне.

Рівні тестування

Unit, Module, Component testing – тестування окремих компонентів ПЗ. Компонент – найменший елемент, який може бути протестовано окремо. Зазвичай проводиться розробниками в кодї.

Integration testing - окремі модулі (компоненти) об'єднуються та тестуються у групі. Зазвичай проводиться розробниками в кодї.

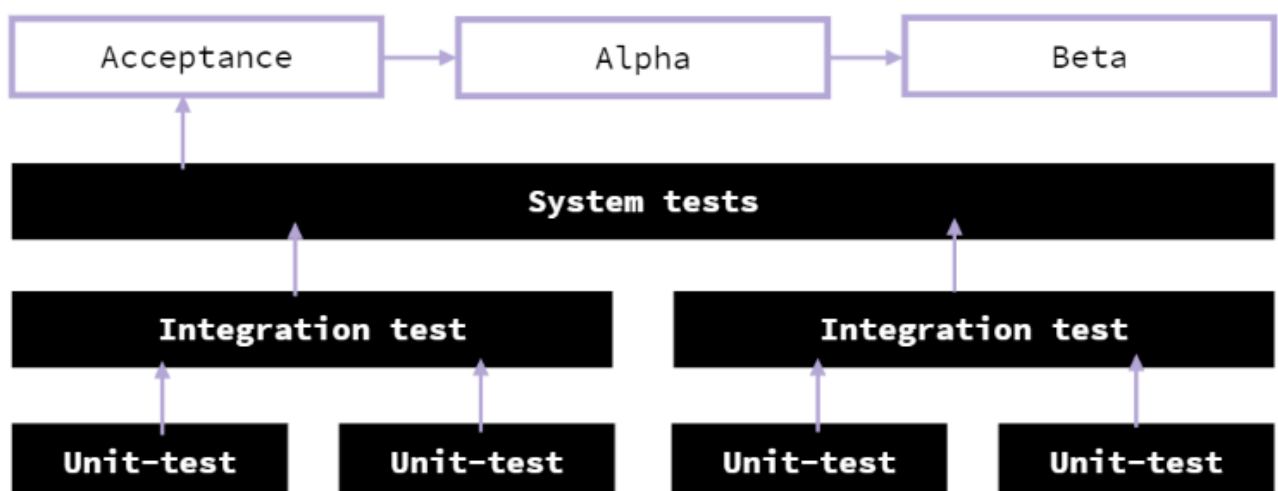
System testing – тестування, яке виконується на повній, інтегрованій системі щодо відповідності всієї системи вихідним вимогам. Це тестування чорної скриньки, яке проводиться тестувальниками.

Acceptance testing – це останнє тестування перед релізом, робиться для перевірки готовності ПЗ виконувати завдання та функції, поставлені під час розробки. До нього відносяться **Альфа** та **Бета**-тестування.

Alpha-testing - тестування всередині компанії командою розробки (програмісти, тестувальники, бізнес аналітики).

Beta-testing – тестування потенційними користувачами.

End-to-end, E2E – ретельне тестування на основі сценаріїв «від початку до кінця» з усіма задіяними функціями.



Типи тестування

Functional testing – це тестування ПЗ з метою перевірки реалізованості функціональних вимог, тобто можливості ПЗ у певних умовах вирішувати завдання, необхідних користувачам.

Non-functional testing – аналіз атрибутів якості компонента чи системи, які не належать до функціональності, тобто перевірка «як працює система».

- **Load testing** – оцінка поведінки системи при зростаючому навантаженні, а також для визначення навантаження, яке здатні витримати компонент або система.
- **Scalability testing** – тестування програмного забезпечення для вимірювання можливостей масштабування.
- **Volume testing** – тестування, у якому система випробовується великих обсягах даних.
- **Stress testing** – вид тестування продуктивності, що оцінює систему на граничних значеннях робочих навантажень або поза ними.
- **Recovery testing** - тестування на якість відновлення системи після падіння.
- **Installation testing** – тестування, спрямоване на перевірку успішного встановлення та налаштування, оновлення або видалення програми.
- **UI testing** – перевірка вимог до інтерфейсу користувача.
- **Usability testing** – перевірка того, наскільки легко кінцевий користувач може зрозуміти і освоїти інтерфейс. Звичайних користувачів із цільової аудиторії запрошують попрацювати з продуктом і спостерігають, як вони виконують завдання та з якими складнощами стикаються у процесі.
- **Localization testing** – перевірка адаптації програмного забезпечення для нового місця експлуатації (наприклад, при зміні мови).
- **Security testing** – тестування програмного продукту для аналізу ризиків, пов'язаних із забезпеченням цілісного підходу до захисту програм, атак хакерів, вірусів, несанкціонованого доступу до конфіденційних даних.
- **Reliability testing** – тестування здатності програми виконувати свої функції у заданих умовах протягом заданого часу.

Типи тестування

Тестування пов'язане із змінами проводиться після виправлення виявлених у процесі тестування помилок та недоліків. Головне завдання – підтвердити усунення проблеми.

- **Re-testing або Confirmation testing** - виправлено дефект чи ні.
- **Regression Testing** - чи не пошкодила зміна в коді інші функції продукту.
- **Smoke Testing** - перевірка критично важливих функцій та стабільності системи загалом перед більш ретельним тестуванням.
- **Sanity Testing** - загальний стан системи у деталях. Доказ працездатності конкретної функції. Часто використовують для перевірки внаслідок внесених змін.
- **Sanity** орієнтоване на глибинне дослідження певної функції, а **Smoke** - на тестування великої кількості функціонала за найкоротші терміни.
- **Build Verification Test** - стабільність системи загалом, перевірка критично важливих функцій конкретного складання. Визначення відповідності версії ПЗ критеріям якості перед початком тестування. Є аналогом димового тестування, але може проникати глибше.

Процес тестування перед релізом також називають **регресією**. Вона відбувається приблизно за місяць до релізу. В цей час у розробників починається **code freeze**, тобто вони не можуть розробляти новий функціонал, тільки виправляють баги. А тестувальники в цей час мусять перевірити якнайбільше тестів на предмет нових дефектів.

Часто поняття **Regression Testing** використовують як збірну назву для всіх видів тестування, пов'язаних зі змінами.

Техніки тест дизайну

Тест-дизайн – це етап процесу тестування ПЗ, на якому проектується та створюються тест-кейси відповідно до визначених раніше критеріїв якості та цілей тестування.

Еквівалентне розділення поділяє дані на групи (класи еквівалентності), які обробляються схожим способом.

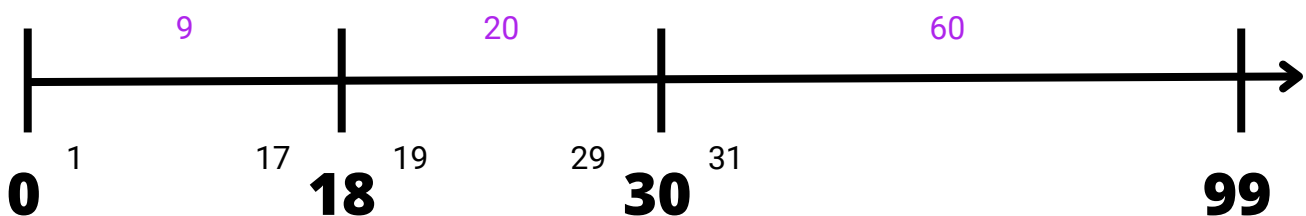
Будь-який тест, виконаний з одного й того ж класу еквівалентності, призведе до такого самого результату, як і виконання решти тестів з цього ж класу.

Граничне значення – це значення, яке знаходиться на межі класів еквівалентності.

Аналіз граничних значень – техніка, що перевіряє поведінку системи чи окремого модуля на граничних значеннях вхідних даних.

Більшість помилок виникає саме на межах між класами еквівалентності. Тобто, тестувальнику насамперед важливо перевірити переходи на стику кордонів кожного класу.

Приклад. Текстове поле в програмі приймає введення віку користувача. Значення від 18 до 30, включаючи 18 і 30, будуть прийняті системою. Використовуючи техніку Граничних значень та Еквівалентного поділу, які тести необхідно написати?



Зверху відображено 3 тести еквівалентного розбиття (9 і 60 років як невалідні тести, 20 років - валідний тест).

Знизу – граничні значення.

Якщо використовувати 2-point граничні значення, то відповідь - 17, 18, 30, 31.

Якщо використовувати 3-point, то відповідь - 17, 18, 19, 29, 30, 31.

Техніки тест дизайну

Decision Table (Таблиці прийняття рішень) – техніка, що допомагає наочно зобразити комбінаторику умов із ТЗ.

По горизонталі описуються **умови**, що впливають на результат. Обов'язково у формі питання. В нижній частині пишуться **дії або наслідки** (опис очікуваного результату).

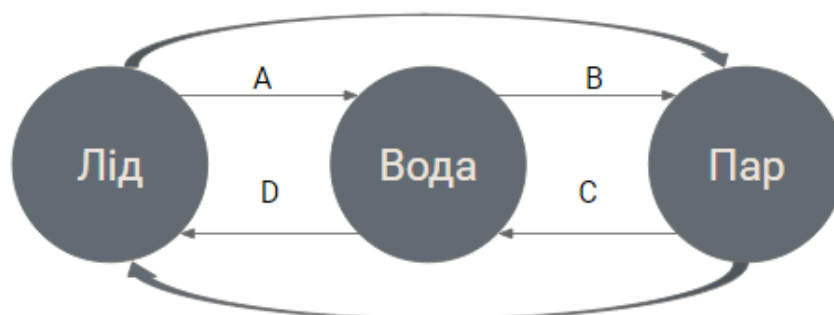
По вертикалі описують **правила** – комбінація вхідних умов, чи простіше різні тести.

ТПР використовують, коли є проблеми із документацією або за вимог, що постійно змінюються.

0 використовується для позначення false (не правда),
1 використовується для позначення true (істина).

| | 1 | 2 | 3 | 4 |
|-----------------|------|------|------|------|
| Формат jpg? | 0 | 0 | 1 | 1 |
| Розмір до 1 Гб? | 0 | 1 | 0 | 1 |
| Результат | Fail | Fail | Fail | Pass |

- **Діаграма переходу станів** показує початковий і кінцевий стан системи, а також описує переходи між станами.
- Діаграма переходу станів показує лише валідні переходи.
- Діаграма складається з пар переходів між двома станами.
- Якщо переходу між двома станами немає, то перехід вважається НЕвалідним.



| Test Case ID | TC01 | TC02 | TC03 | TC04 |
|-----------------|------|------|------|------|
| Початковий стан | Лід | Вода | Вода | Пар |
| Перехід | A | D | B | C |
| Фінальний стан | Вода | Лід | Пар | Вода |

Техніки тест дизайну

Pairwise testing (Метод попарного тестування) заснований на ідеї, що переважна більшість багів виявляється тестом, що перевіряє або один параметр, або поєднання двох.

Помилки, причиною яких стали комбінації трьох і більше параметрів, як правило, значно менш критичні.

Використовуючи метод попарного тестування, ми не тестуємо всі можливі поєднання, а складаємо тести так, щоб кожне значення параметра хоча б один раз поєднувалося з кожним значенням інших параметрів, що тестуються.

Таким чином, метод суттєво скорочує кількість тестів, а отже, і час тестування.

ПЗ для автоматичного формування перевірок попарного тестування: **allpairs**, **PICT**, **Pairwise online tool**, **VPTag**, **ACTS** та ін.

Error guessing (Вгадування помилок) – це спосіб запобігання помилкам, дефектам та відмовам, заснований на знаннях тестувальника, що включають:

- Історію роботи програми в минулому.
- Найбільш ймовірні типи дефектів, які допускаються під час розробки.
- Типи дефектів, які були виявлені у подібних додатках.

У передбаченні помилок немає чіткої та логічної схеми, яка б дозволила нам скласти тест-кейси. Навпаки, ця техніка ґрунтується на досвіді тестувальника та на його вмінні думати креативно та деструктивно.

Прикладами тестів для вгадування помилок є:

- Ділення на 0
- Вибір дати народження в майбутньому
- Вибір неіснуючої дати (30 лютого)
- Введення пробілів у текстові поля
- Натискання кнопки надсилання без введення значень
- Завантаження файлів, що перевищують максимальні обмеження

Техніки тест дизайну

Exploratory Testing (Дослідницьке тестування)

- Дослідницьке тестування перевіряє частину програми, яку необхідно вивчити
- У дослідному тестуванні створюються неформальні тести, вони виконуються, створюється звіт і все це робиться у процесі тестування
- Тест кейси не вимагають підготовки та виконуються у рандомній або Ad-Hoc манері
- Виконання тестів обмежено проміжками часу від 30 до 120 хвилин, залежно від розміру програми
- Виконання тестів називають сесіями тестування у проміжках часу

Ad-hoc Testing - повністю неформалізований підхід, у якому не передбачається використання ні тест-кейсів, ні чек-листів, ні сценаріїв. Тестувальник спирається на свою інтуїцію та досвід для спонтанного виконання з продуктом дій, які, як він вважає, що можуть виявити помилку.

Exploratory та Ad-hoc тестування - це різні техніки дослідження продукту з різним ступенем формалізації, різними завданнями. Різниця між Ad-hoc та Exploratory тестуванням у тому, що, теоретично, Ad-hoc може провести будь-хто. А для проведення Exploratory необхідна майстерність та володіння певними техніками.

Можна сказати, що Ad-hoc тестуванням займаються бета-тестувальники, які добровільно зголосилися використовувати продукт та повідомляти про помилки. Вони саме поняття не мають ні про техніки тестування, ні про його методи та принципи.

Monkey Testing – це метод тестування ПЗ, при якому тестувальник вводить будь-які випадкові вхідні дані без заздалегідь визначених тестових випадків і перевіряє поведінку програми, незалежно від того, дає вона збій чи ні. Тобто, тестуємо без мети, без плану. Просто тикаємося скрізь із наміром щось зламати. Як мавпочка.

Фронтенд / Бекенд тестування

Фронтенд – це розробка інтерфейсу користувача та функцій, які працюють на клієнтській стороні веб-сайту або програми. Це все, що бачить користувач, відкриваючи веб-сторінку і з чим він взаємодіє. Фронтенд обмінюється з бекендом інформацією через запити.

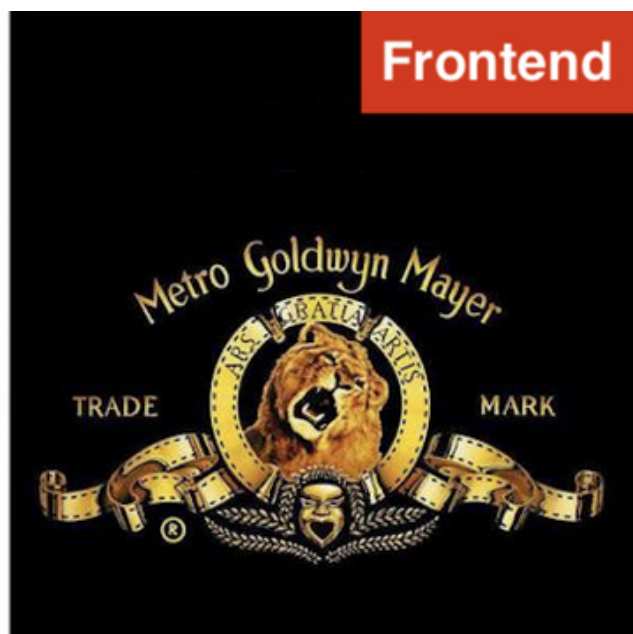
Бекенд – код програми, який виконується не на пристрої користувача, а на віддаленому сервері.



Backend

Back-end testing

- Включає тестування баз даних та бізнес-логіки.
- Тестер повинен мати великий досвід роботи з базами даних та концепціями мови структурованих запитів (SQL).
- Важливо для перевірки взаємоблокування, пошкодження даних, втрати даних тощо.
- Широко використовується тестування бази даних: SQL-тестування та API-тестування.



Frontend

Front-end testing

- Завжди виконується у графічному інтерфейсі.
- Тестер повинен бути поінформований про вимоги бізнесу, а також використання інструментів середовищ автоматизації.
- GUI використовується для тестування.
- Не потрібно зберігати інформацію в базі даних.
- Важливо перевірити загальну функціональність програми.

Фронтенд / Бекенд термінологія

DevTools – це набір інструментів, вбудованих у браузер, для створення та налагодження сайтів. З їх допомогою можна переглядати вихідний код сайту, налагоджувати роботу frontend: HTML, CSS та JavaScript. Також DevTools дозволяє перевіряти мережевий трафік, швидкодію сайту та багато іншого. За допомогою режиму емуляції DevTools дозволяє переглядати веб-сторінки у мобільному вигляді.

Сніффер – це інструмент, який дозволяє перехоплювати, аналізувати та модернізувати всі запити, які через нього проходять. За допомогою нього можна витягти з потоку будь-які відомості або створити відповідну відповідь сервера.

Кеш (Cache) – це копії завантажених веб-сторінок. Якщо повторно заходити на сайт, його завантаження відбувається не з інтернету, а з жорсткого диска, де зберігаються тимчасові файли. Це пришвидшує роботу браузера. Веб-сторінки можуть відображатися некоректно через те, що в них були внесені зміни, а браузер продовжує використовувати застарілі дані з кешу. Щоб забезпечити коректність відображення веб-сторінок перед проведенням тестування, потрібно обов'язково очищати кеш браузера.

Cookie – тимчасові файли, що зберігаються на жорсткому диску комп'ютера користувача. Куки зберігають налаштування сайтів, які відвідував користувач. Найпоширеніша функція – збереження паролів, що дозволяє не вводити комбінацію логін + пароль щоразу при вході на сайт.

Мови HTML та CSS призначені для верстки сайтів.

Мова PHP потрібна для програмування сайту, за його допомогою можна, наприклад, зробити реєстрацію користувачів.

Мова JavaScript потрібна для того, щоб «оживити» сайт: наприклад, зробити картинки, що змінюються (слайдер).

HyperText Markup Language

Мова HTML (HyperText Markup Language) - мова гіпертекстової розмітки. Базова мова для створення веб-сторінок.

HTML складається з: тегів, атрибутів, значень атрибутів, вмісту елемента.

HTML-теги – це спеціальні команди для браузера. Вони кажуть, наприклад, що слід вважати заголовком сторінки, а що абзацом.

Назва тега може складатися з англійських букв та цифр. Приклади тегів: `<h1>`, `<p>`, ``.

Теги зазвичай пишуться парами - відкриваючий і закриваючий теги. Різниця між ними в тому, що в закриваючому тегу після куточка стоїть слеш /. Приклад: `<p>.....</p>`.

Бувають теги, які не потрібно закривати, наприклад `
` або ``.

Атрибути - спеціальні команди, які розширюють дію тега. Атрибути розміщуються всередині тега, в такому форматі:

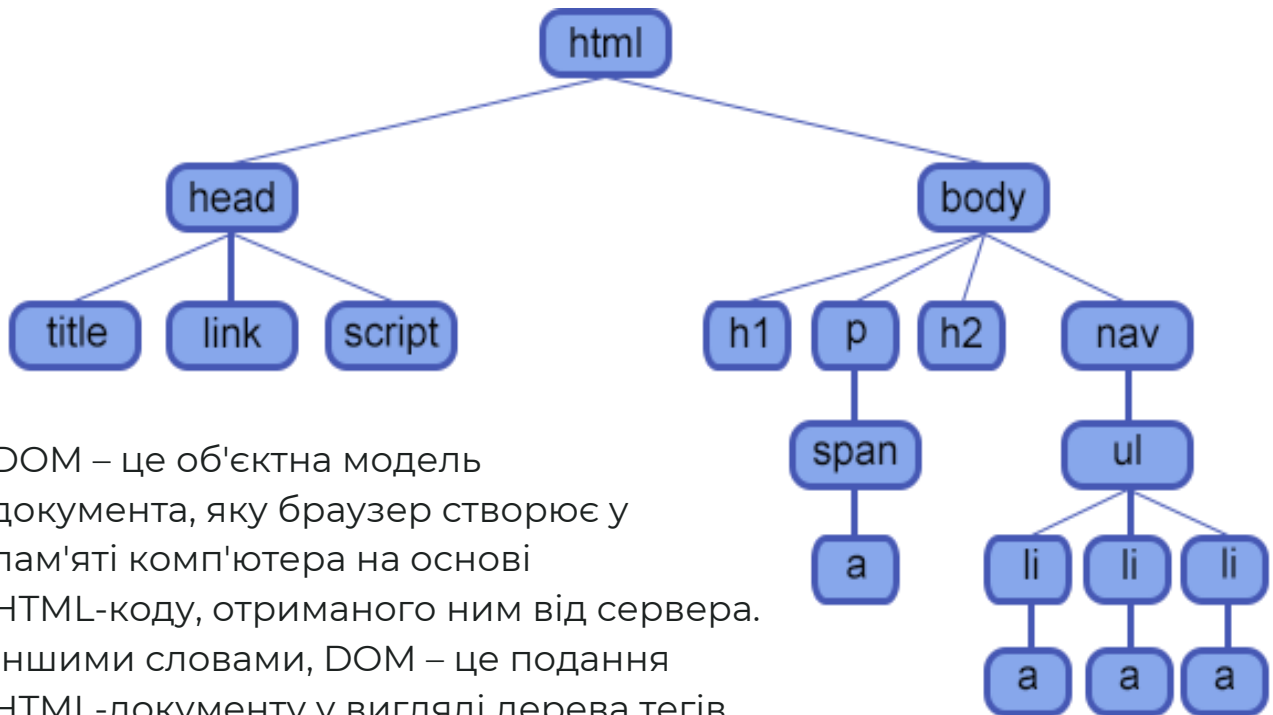
```
<тег атрибут1="значення" атрибут2="значення">
```

Лапки можуть бути будь-якими – одинарними або подвійними, допустимо взагалі їх не ставити, якщо значення атрибута складається з одного слова.

Теги, які є в більшості HTML-файлів:

- `<html>` - Вказує, що це HTML код.
- `<head>` – містить технічну інформацію, наприклад, заголовок сторінки, стилі та скрипти, що підключаються і т.д.
- `<title>` (заголовок) – містить заголовок HTML-документа, який відображається у вкладці браузера.
- `<body>` – вміст документа, який бачить користувач.
- `<meta>` – задає опис вмісту сторінки та ключові слова, автора HTML-документу та ін. властивості метаданих.

DOM (Document Object Model)



DOM – це об'єктна модель документа, яку браузер створює у пам'яті комп'ютера на основі HTML-коду, отриманого ним від сервера. Іншими словами, DOM – це подання HTML-документу у вигляді дерева тегів.

Cascading Style Sheets

CSS розшифровується як **Cascading Style Sheets**, що в перекладі означає «Каскадні Таблиці Стилів».

Мова CSS – це мова розмітки, яка використовується для візуального оформлення веб-сайтів, вона розширює можливості HTML.

CSS дозволяє швидко змінити візуальне оформлення сайту, не вдаючись до більш складних мов програмування. З його допомогою можна змінити кольори, шрифти, фон, взагалі займатися красою сайту.

На відміну від HTML, який служить для визначення структури та семантики вмісту сайту, CSS відповідає за його зовнішній вигляд та відображення.

Основи Баз Даних

База даних (БД) - це впорядкований набір структурованої інформації або даних, які зазвичай зберігаються в електронному вигляді комп'ютерної системи.

Система управління базами даних (СУБД) – комплекс програм, що дозволяють створити БД та маніпулювати даними (вставляти, оновлювати, видаляти та вибирати).

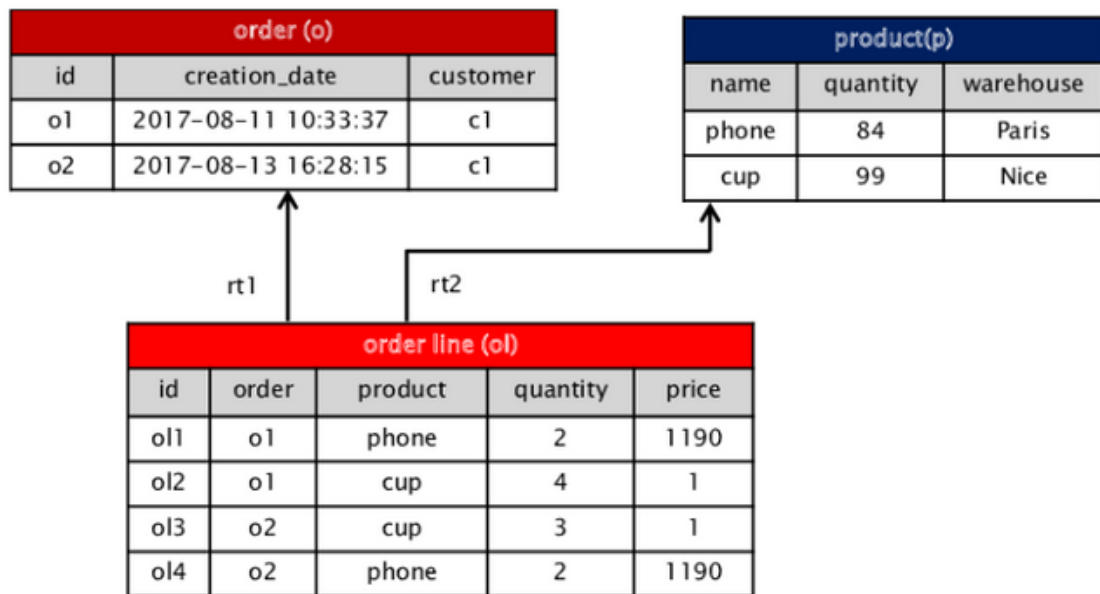
Реляційна БД – це набір даних із зв'язками між ними.

Мова SQL - структурована мова запитів, що використовується для роботи з реляційними БД і допомагає писати команди, що створюють, зчитують, змінюють або видаляють запити у БД.

- Таблиця являє собою набір пов'язаних записів даних і складається з безлічі стовпців і рядків.
- Таблиця складається з впорядкованого набору рядків та стовпчиків.
- Перетином рядка і стовпчика є клітинка (комірка).
- Поле являє собою стовпець таблиці, який призначений для зберігання конкретної інформації про кожний запис в таблиці.
- Рядки (записи) містять всю інформацію про сутність.
- Значення NULL в таблиці — це значення в полі, яке є порожнім, що означає, що поле зі значенням NULL є полем без значення.

| Customers | | | | | |
|-----------|------------|-----------|------------|-----------------------|----|
| | CustomerId | FirstName | LastName | DateCreated | Cl |
| + | 1 | Homer | Simpson | 13/06/2014 3:33:37 PM | |
| + | 2 | Peter | Griffin | 13/06/2014 9:09:56 PM | |
| + | 3 | Stewie | Griffin | 13/06/2014 9:16:07 PM | |
| + | 4 | Brian | Griffin | 13/06/2014 9:16:36 PM | |
| + | 5 | Cosmo | Kramer | 13/06/2014 9:16:41 PM | |
| + | 6 | Philip | Fry | 13/06/2014 9:17:02 PM | |
| + | 7 | Amy | Wong | 13/06/2014 9:22:05 PM | |
| + | 8 | Hubert J. | Farnsworth | 13/06/2014 9:22:19 PM | |
| + | 9 | Marge | Simpson | 13/06/2014 9:22:37 PM | |
| + | 10 | Bender | Rodriguez | 13/06/2014 9:22:52 PM | |
| + | 11 | Turanga | Leela | 13/06/2014 9:23:37 PM | |
| * | (New) | | | 15/06/2014 9:00:01 PM | |

Команди SQL



Ключ СУБД – це атрибут чи набір атрибутів, що допомагає ідентифікувати рядок у таблиці.

Primary key – поле, кожен елемент якого однозначно визначає запис таблиці. Використовується для визначення об'єкта.

Foreign key – це стовпець чи поєднання стовпців, яке застосовується для примусового встановлення зв'язок між даними однієї БД.

Команди, що працюють зі структурою БД:

- **CREATE** – створити. Наприклад, **CREATE TABLE** – створити таблицю або **CREATE USER** – створити користувача.
- **ALTER** – модифікувати. Цей запит використовується при внесенні змін до самої БД або її частини.
- **DROP** – Видалити. Запит відноситься до БД та її частин.

Команди, що працюють з даними:

- **SELECT** – вибір даних.
- **INSERT** – вставлення нових даних.
- **UPDATE** – оновлення даних.
- **DELETE** – видалення даних.
- **MERGE** – злиття даних.

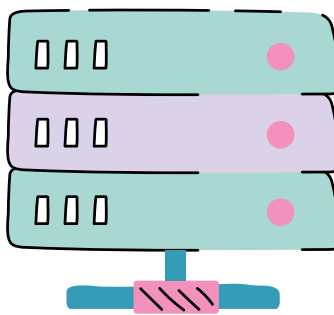
Клієнт-серверна архітектура

- **Клієнт** – комп'ютер на стороні користувача, який відправляє запит до сервера для надання інформації або виконання певних дій.
- **Сервер** – більш потужний комп'ютер або обладнання, призначене для вирішення певних завдань з виконання програмних кодів, виконання сервісних функцій за запитом клієнтів, надання користувачам доступу до певних ресурсів, зберігання інформації і баз даних.
- **Мережевий протокол** - це набір правил, за якими клієнт та сервер взаємодіють.



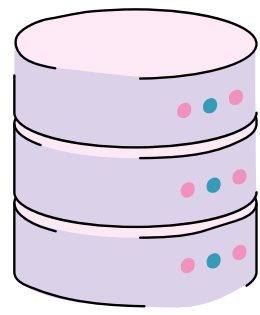
Клієнт

Клієнт відправляє HTTP запит до API, а у відповідь отримує HTTP відповідь та обробляє її.



Сервер

Клієнт спілкується із сервером через API. Сервер приймає HTTP запит, обробляє його та видає клієнту HTTP відповідь.



База даних

База даних приймає SQL запит від сервера та видає відповідь назад.

На клієнт-серверній архітектурі побудовані всі сайти та інтернет-сервіси, а також низка десктоп-програм, які передають дані по інтернету.

За рівнями архітектура буває:

- Дворівнева - містить тільки Клієнт та Сервер, без БД.
- Трирівнева - включає взаємодію з БД.
- Багаторівнева - запит клієнта обробляється відразу кількома серверами. Як приклад можна навести будь-яку сучасну СУБД.

Основи API

API (Application programming interface) – опис способів і правил, якими одна комп'ютерна програма може взаємодіяти (спілкуватися і обмінюються даними) з іншого.

- **Зовнішні (сторонні) API** - доступні для всіх, включаючи сторонніх розробників програмного забезпечення. Прикладом є Google карти, реєстрація через соц. мережі. Тобто розробляти їх окремо не треба, це готові рішення для використання.
- **Внутрішня API** - функції, що розробляються під потреби конкретного сайту. Приклад – пошук сайту з фільтрацією за якимись критеріями.

На клієнт-серверній архітектурі побудовані всі сайти та інтернет-сервіси, а також низка десктоп-програм, які передають дані по інтернету.

За рівнями архітектура буває:

- Дворівнева - містить тільки Клієнт та Сервер, без БД.
- Трирівнева - включає взаємодію з БД.
- Багаторівнева - запит клієнта обробляється відразу кількома серверами. Як приклад можна навести будь-яку сучасну СУБД.

HTTP (HyperText Transfer Protocol) - протокол клієнт-серверної взаємодії, що ініціює запити до сервера самим одержувачем. Обмін повідомленнями відбувається за схемою «запит-відповідь».

На даний момент завдяки протоколу HTTP забезпечується робота інтернету. Зазвичай програма здійснює доступ до веб-ресурсів через веб-браузер. Для ідентифікації ресурсів HTTP використовує URI.

Найпопулярніший тип **URI** – це **URL (Uniform Resource Locator)**, яку також називають веб-адресою. Крім ідентифікації ресурсу, URL надає інформацію про місцезнаходження цього ресурсу в Інтернеті.

HTTP – це відкритий протокол передачі між браузером і сервером. **HTTPS** – той самий HTTP, але з доданими методами шифрування даних та перевірки безпеки, тобто захищений протокол.

HTTP request – це повідомлення, що надсилаються клієнтом, щоб ініціювати реакцію з боку сервера.

HTTP response – це повідомлення, яке надсилає сервер у відповідь на запит.

Структура HTTP запиту:

- **Request line** - версія протоколу HTTP та URL, до якого має звернутися сервер.
- **HTTP Method** – команда, за якою сервер визначає, що потрібно робити. Зазвичай це коротке англійське слово, записане великими літерами.
 - GET – використовується, коли хочемо отримати якісь дані з сервера;
 - POST – використовується, коли нам потрібно створити новий об'єкт або внести зміни;
 - PUT – використовується, якщо нам треба внести зміни до існуючого об'єкта;
 - DELETE – використовується, коли потрібно видалити об'єкти чи сутності;
- **Headers (Заголовки)** - параметри, які генеруються автоматично, але туди можна додати і свої (наприклад, для авторизації).
- **Request Body (Тіло запиту)** - інформація, яку передав браузер під час запиту сторінки. Тіло не є обов'язковим параметром. Замість тіла можуть бути **параметри** (пари ключ-значення) для передачі інформації.

Структура HTTP відповіді:

- **Status code** - частина відповіді сервера, яка інформує клієнта про результат запиту. Складається він із трьох цифр, перша з яких вказує на клас стану.
 - 100 - інформаційні;
 - 200 - успіх;
 - 300 - перенаправлення;
 - 400 - помилка на стороні клієнта;
 - 500 - помилка на стороні сервера;
- **Response headers (Заголовки)** - службова інформація.
- **Response body (Тіло повідомлення)** - дані, які надсилає сервер у відповідь на запит.

Тестування API

- **Тестування API** – це тип тестування, який зосереджений лише на рівні бізнес-логіки архітектури ПЗ.
- **Метою тестування API** є перевірка функціональності, надійності, продуктивності та безпеки програмних інтерфейсів. Тестування API відноситься до інтеграційного тестування, а значить, під час нього можна відловити помилки взаємодії між модулями системи або між системами. Також розглядаються питання безпеки.
- **Веб-сервіси** - це спосіб зв'язку (обмін даними) між двома електронними пристроями (додатками) по мережі.

Протоколи реалізації веб-сервісів:

REST

- Representational State Transfer
- REST - архітектурний стиль
- Передаємо Json, але можемо XML і текст
- Використовує HTTP (має методи запитів, тіло, статус коди, заголовки)
- Більш простий для реалізації

SOAP

- Simple Object Access Protocol
- SOAP - формат обміну даними
- Передаємо XML
- Використовує WSDL (Web Service Description Language) - мову опису веб-сервісів та доступів до них, на основі XML
- Більш безпечний

Тестування API має ряд переваг перед UI:

- Точне розуміння, де відбувається помилка та чим вона викликана.
- Витрачається менше часу на підготовку тестових даних.
- Можливе виконання тестів на великих обсягах даних з прийнятною швидкістю.
- Можна розпочати тестування на ранніх етапах, коли ще немає інтерфейсу.

Мобільне тестування

Мобільне тестування – процес тестування програм для сучасних мобільних пристроїв на функціональність, зручність використання, продуктивність та ін.

Особливості тестування мобільних додатків

- Тестування взаємодії користувача – зручності роботи з додатком: свайпи, тапи, скроли тощо.
- Тестування сумісності - установка на різні ОС, платформи, на різних моделях, перевірка на різних дозволах тощо.
- Тестування підключення – перевірка на різних типах підключення (wi-fi, мобільна мережа), перемикання типів та офлайн робота.
- Тестування продуктивності – витік пам'яті, стабільність роботи за великої кількості користувачів тощо.
- Тестування локалізації – перевірка розміщення локалізованого (перекладеного) тексту на екрані, формату дат тощо.

Adaptive Design (AWD) – проектування сайту з декількома статичними макетами для різних типів пристроїв (**мобільні пристрої, планшети, настільні комп'ютери**). Тобто макети завантажуються за певних розмірів вікна браузера пристрою, базуючись на контрольних (переломних) точках.

Responsive Design (RWD) - проектування сайту з певними значеннями властивостей, наприклад гнучка сітка макета, які дозволяють **одному макету працювати на різних пристроях**.

Кроссбраузерність – це здатність веб-ресурсу відображатися однаково та працювати у всіх популярних браузерах без перебоїв у функціонуванні та помилок у верстці, а також з однаково коректною читабельністю контенту.

Кроссплатформенність – здатність програмного забезпечення працювати з кількома апаратними платформами або операційними системами.

Типи мобільних додатків

Нативні додатки розробляються «під платформу», тобто. для конкретної ОС: iOS, Android або Windows. Програма встановлюється на смартфон із оф. магазину. Нативні сервіси можуть працювати незалежно від підключення до інтернету, хоча частина потребує підключення.

Веб-додатки є адаптованими веб-сайтами, які відкриваються через браузер. Користувач не завантажує програму і не зберігає її на своєму пристрої. Одним з найпоширеніших підвидів вважають PWA — прогресивні веб-програми, які, по суті, є нативними програмами всередині браузера.

Гібридні додатки – додаток, який поєднує в собі елементи нативного та веб-додатку. Додаток необхідно завантажувати на пристрій (як нативні), але написаний за допомогою HTML, CSS і JavaScript.

Instant Apps – це програми, які можна завантажити та запустити без необхідності проходження повного циклу установки. Доступні лише для Android.

Чек-ліст тестування мобільних додатків

- Функціональне тестування
- Тестування сумісності з іншими версіями ОС, різними оболонками та сторонніми сервісами (наприклад, оплата), апаратним забезпеченням пристрою, підключення зовнішніх пристроїв, тестування переривань.
- Тестування безпеки. Тестування дозволів (доступ до камери/мікрофону/галереї/і т.д.). Тестування передачі даних користувача (паролі) тощо.
- Тестування локалізації та глобалізації
- Тестування зручності використання
- Стресове тестування
- Кросс-платформне тестування
- Тестування продуктивності

Тестування мобільних додатків

Тестування переривань – це тестування реакції мобільного додатка на переривання та повернення до свого попереднього стану.

Види переривань:

- Низький заряд батареї.
- Вхідний дзвінок.
- Вхідні смс.
- Вхідне оповіщення з іншого мобільного додатка.
- Підключений для заряджання.
- Вимкнено від заряджання.
- Пристрій вимкнено.
- Нагадування про оновлення програми.
- Аварійна сигналізація.
- Втрата підключення до мережі. Відновлення підключення до мережі.

Симулятори і Емулятори

Емулятор – імітує програмну (софт) та апаратну (залізо) частину пристрою. Мета роботи на емуляторі – створення точної моделі пристрою та повна перевірка коректності роботи програми на цьому пристрої.

Симулятор – імітує програмне забезпечення пристрою. Мета роботи на симуляторі – зрозуміти, як працювати в оригінальній програмі, вивчити її інтерфейс, а також зрозуміти, як реагуватиме програма на дії користувачів.

Емулятори не можуть імітувати проблеми з батареєю, вхідні переривання, проблеми з датчиками GPS та освітлення, команди жестами, проблеми з сенсорним екраном, перенесення кольорів.

Генератори даних

- Mockaroo.com (mockaroo.com) Генерує тестові дані людини (ім'я, номер картки та інше)
- Bugmagnet – плагін для Chrome та Firefox. Генерує різні дані для заповнення полів.
- Generatedata.com (generatedata.com) Генерує дані користувача та формує запити.
- Генератор зображень (placeimg.com)
- Генератори тимчасових поштових скриньок (temp-mail.org)
- Генератор особи (fakenamegenerator.com/advanced.php)

Інструменти для чек-лістів

- Testpad (ontestpad.com) – інструмент для складання плану тестування та контролю за допомогою списків. Комфортний та гнучкий у роботі.
- Sitechco (sitechco.ru) – онлайн-сервіс для ведення чеклістів, що дозволяє зберігати результати, переглядати звіти та статистику. Велике достоїнство Sitechco - можливість інтеграції з Jira Cloud.
- Teamsuccess (teamsuccess.io) – сервіс, в якому перераховані деякі перевірки, які стануть у нагоді при тестуванні та допоможуть структурувати ідею.
- Google-таблиці дуже корисний та ефективний інструмент у вмілих руках.