

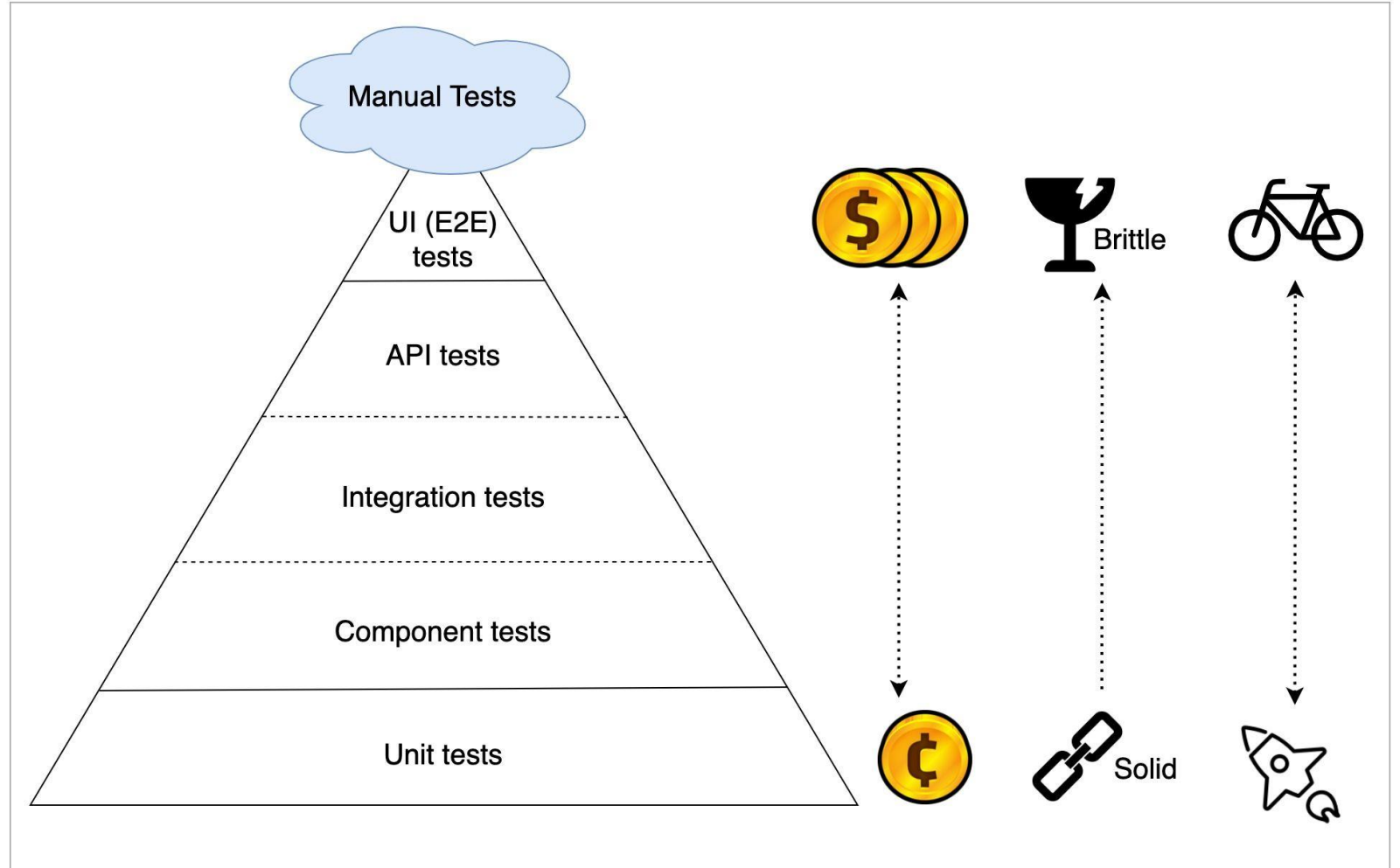
PET MANAGER

TESTING PLAN



Metodologia testowania

- Zastosowanie piramidy testów.
- Uwzględnienie ryzyk i niepewności (chmura niepewności).
- Testy jednostkowe, integracyjne, akceptacyjne i wydajnościowe jako fundament planu testowania.



Testy jednostkowe



Testy, których przeprowadzi się najczęściej na prostych funkcjach i metodach. Każdy dodany feature będzie musiał być przetestowany.



Przykłady: Dodanie nowego zwierzęcia do aplikacji, funkcja przypominania o karmieniu, tworzenie konta itp.



Narzędzia: JUnit lub Unittest

Testy integracyjne



Sprawdzenie współpracy między modułami aplikacji.



Przykład: Tworzenie kont przy pomocy Google (Google SSO), komunikacja z bazą danych, API od innych aplikacji lub serwisów. To wszystko musi zostać przetestowane



Narzędzia: Postman, Cypress lub tworzenie mock-ów, stub-ów, fake-ów.

Testy akceptacyjne

Weryfikacja, czy aplikacja spełnia wymagania użytkowników.

Przykład: Test poprawności powiadomień push, wyszukiwanie zwierząt, formalne zatwierdzenie logowania użytkownika.


Narzędzia: Selenium, Appium.

Testy UI/E2E

Sprawdzenie pełnego przepływu działania aplikacji.
Generalne zasymulowanie użytkowania aplikacji



Przykład: Rejestracja użytkownika, dodanie zwierzęcia, ustawienie powiadomienia. Dodanie maksymalnej ilości zwierząt.



Narzędzia: Najlepszym wyborem jest Appium, ponieważ jest to aplikacja mobilna, aczkolwiek można wykorzystać Cypress

Skorzystanie z piramidy i tworzenie względem niej scenariuszy testowania zależnie od rodzaju testów jest zazwyczaj standardową procedurą w tworzeniu planu testowania aplikacji, lecz bardzo istotne jest aby wykonać to zgodnie z budżetem projektu by nie doszło do długu technologicznego.



Źródło: <https://martinfowler.com/bliki/TechnicalDebtQuadrant.html>

Chmura Niepewności

Potencjalne ryzyka:

Nieoczekiwane błędy integracji:

- Przykład: Powiadomienia push nie są wysyłane w określonym czasie.
- **Wykrycie:** Regularne testy integracyjne. Refactor.

Wydajność:

- Przykład: Aplikacja zawiesza się przy dużej liczbie użytkowników.
- **Mitigacja:** Testy obciążeniowe i wydajnościowe. Odpowiednie korekty serwerowe.

Nieprzewidywalne zachowania UI:

- Przykład: Problemy z responsywnością interfejsu na urządzeniach mobilnych.
- **Mitigacja:** Testy UI na wielu urządzeniach i systemach. Poprawa funkcjonalności

W fazie developmentu, koniecznie trzeba będzie zadbać o **organizację testów oraz ich automatyzację**.

- **Proces:**
- Testy jednostkowe uruchamiane przy każdym commitcie (CI/CD).
- Testy integracyjne i akceptacyjne uruchamiane cyklicznie.
- Testy wydajnościowe realizowane przed każdą dużą aktualizacją.

Narzędzia: Jenkins, CircleCI do automatyzacji procesu testowania. Oczywiście odpowiednia konfiguracja pliku yml przy pomocy github actions.

PODSUMOWANIE



Piramida testów zapewnia efektywność i stabilność.



Regularne testy redukują ryzyko błędów produkcyjnych.



Automatyzacja testów zwiększa szybkość dostarczania aplikacji.