Testowanie Oprogramowania

Patrycja Hajduga, Radosław Gala, Michał Tomala

Politechnika Śląska 2018

1 Podstawy
Trochę teorii

Co wiemy o testowaniu?

- **Testowanie** jest to zbiór czynności wykonywanych z intencją wykrycia jak największej liczby błędów. Jest kontrolą jakości oprogramowania na każdym etapie jego powstawania. Celem jest badanie poprawności aplikacji oraz wczesne wykrycie ewentualnych błędów, co ma bezpośrednie przełożenie na koszty.
- **Proces testowania ma dwa główne cele:**
  - weryfikację oprogramowania, czyli sprawdzenie, czy wytwarzane oprogramowanie jest zgodne ze specyfikacją,
  - walidację oprogramowania, która polega na kontroli czy oprogramowanie jest zgodne z oczekiwaniami użytkownika (pozwala uzyskać żądane wyniki).

Co to jest testowanie?

- Za testowanie najczęściej uważa się tylko wykonywanie testów, to jest uruchamianie oprogramowania.
- Czynności testowania występują zarówno przed, jak i po wykonywaniu testów.
- Planowanie i nadzór
- Wybór warunków testowych
- Projektowanie i wykonywanie przypadków testowych
- Sprawdzanie wyników
- Ocena spełnienia kryteriów zakończenia
- Raportowanie procesu testowania i testowanego systemu oraz kończenie i zamykanie testów (po zakończeniu fazy testów)
- Przeglądy dokumentacji i kodu źródłowego oraz analiza statyczna

Po co testujemy?

- Znajdowanie usterek
- Nabieranie zaufania do poziomu jakości
- Dostarczanie informacji potrzebnych do podejmowania decyzji
- Zapobieganie defektom

Definicja błędu

- Oprogramowanie nie robi czegoś co zostało wymienione w jego specyfikacji lub robi coś czego według specyfikacji nie powinno robić
- Oprogramowanie robi coś o czym specyfikacja nie wspomina
- Oprogramowanie nie wykonuje czegoś o czym specyfikacja nie wspomina mimo że powinno to być wymienione jako istotna część systemu.
- Oprogramowanie jest trudne do zrozumienia, powolne lub skomplikowane.

Aksjomaty testowania

- Całkowite przetestowanie programu jest niemożliwe.
- W testowanie wpisane jest ryzyko.
- Test nie dowodzi że błędu nie ma a duża ilość znalezionych błędów świadczy o jeszcze większej ilości błędów, których nie znaleziono.
- Nie wszystkie znalezione błędy zostaną naprawione.
- Paradoks pestycydów - oprogramowanie uodparnia się na testy

Podział testów - funkcjonalne

- Testy modułowe (unit/component tests)
- Testy integracyjne (integration tests)
- Testy systemowe (system tests)
- Testy akceptacyjne (acceptance tests)

2 Podział Testów

Testy jednostkowe (Unit Tests)

- Sprawdzają konkretny obszar - funkcję lub jednostkę kodu.
- Treść testów znajduje się tam gdzie kod samej aplikacji.
- Każdy nowy kod powinien posiadać odpowiednią ilości testów jednostkowych, które go pokrywają.
- Mają ograniczony zakres i nigdy nie odwołują się do innych komponentów czy też modułów systemu.
- Używają symulacji, mocków oraz sztucznych danych.
- Można i powinno się je wykonywać szybko i często.
- Początkowo testy jednostkowe mogą się wydawać kosztowne w napisaniu pod względem czasu, jednak z tego rodzaju inwestycji jest bardzo szybki zwrot.
- Chronią nas przed większymi problemami i dają swobodę w dodawaniu nowej funkcjonalności
- Dają programiście pewność, że kod będzie działać.

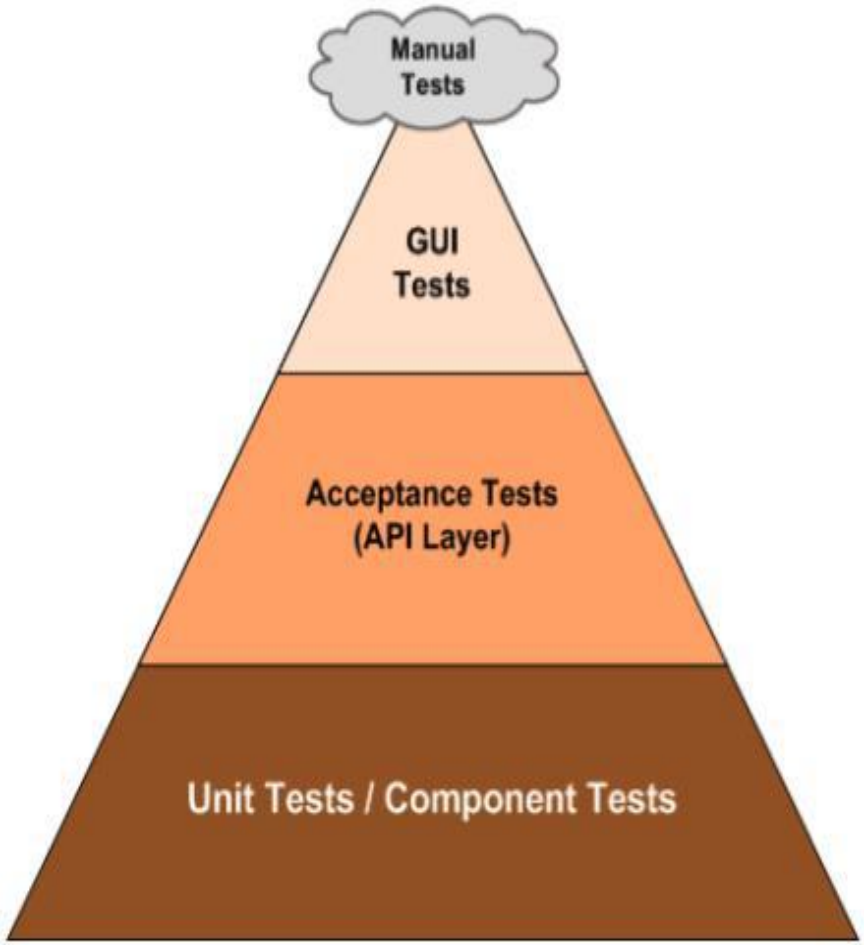
Testy jednostkowe c.d.

- Testy jednostkowe dowodzą że kod działa.
- Za ich pomocą budujesz nisko poziomowy zestaw testów regresyjnych.
- Pozwalają łatwo poprawiać swoją aplikację bez dodatkowego testowania.
- Przyjemniej pisać kod z nimi niż bez nich.
- Demonstrują konkretny postęp.
- Są formą przykładowego kodu.
- Zmuszają programistę do podstawowego planowania przed kodowaniem.
- Redukują koszt błędów.
- Działają lepiej niż inspekcja kodu.
- Pozytywnie wpływają na design aplikacji

Testy integracyjne (Integration Tests)

- Podobne w założeniu jak testy jednostkowe natomiast obejmują większa porcję systemu i nie ograniczają się tylko do małych fragmentów kodu.
- Wykonywane są w celu wykrycia błędów w interfejsach i interakcjach pomiędzy modułami lub systemami.
- Skrajnym przypadkiem tego typu testów są testy Workflow, które zwykle powstają w momencie gdy cały kod aplikacji jest już gotowy i przyjmują scenariusz przejścia przez cały system.
- Pomagają bardzo wcześnie odkrywać błędy w algorytmach i logice na poziomie modułów.

Piramida Testów



Założenia:

- Duża liczba bardzo małych testów jednostkowych - oprzeć fundamenty na prostych testach.
- Mniejsza liczba testów funkcjonalnych testujących najważniejsze komponenty.
- Jeszcze mniejsza liczba testów sprawdzających całą aplikację i workflow.
- Zawsze będą testy, które nie mogą lub nie powinny być zautomatyzowane - stąd konieczność testów manualnych.

Zasady:

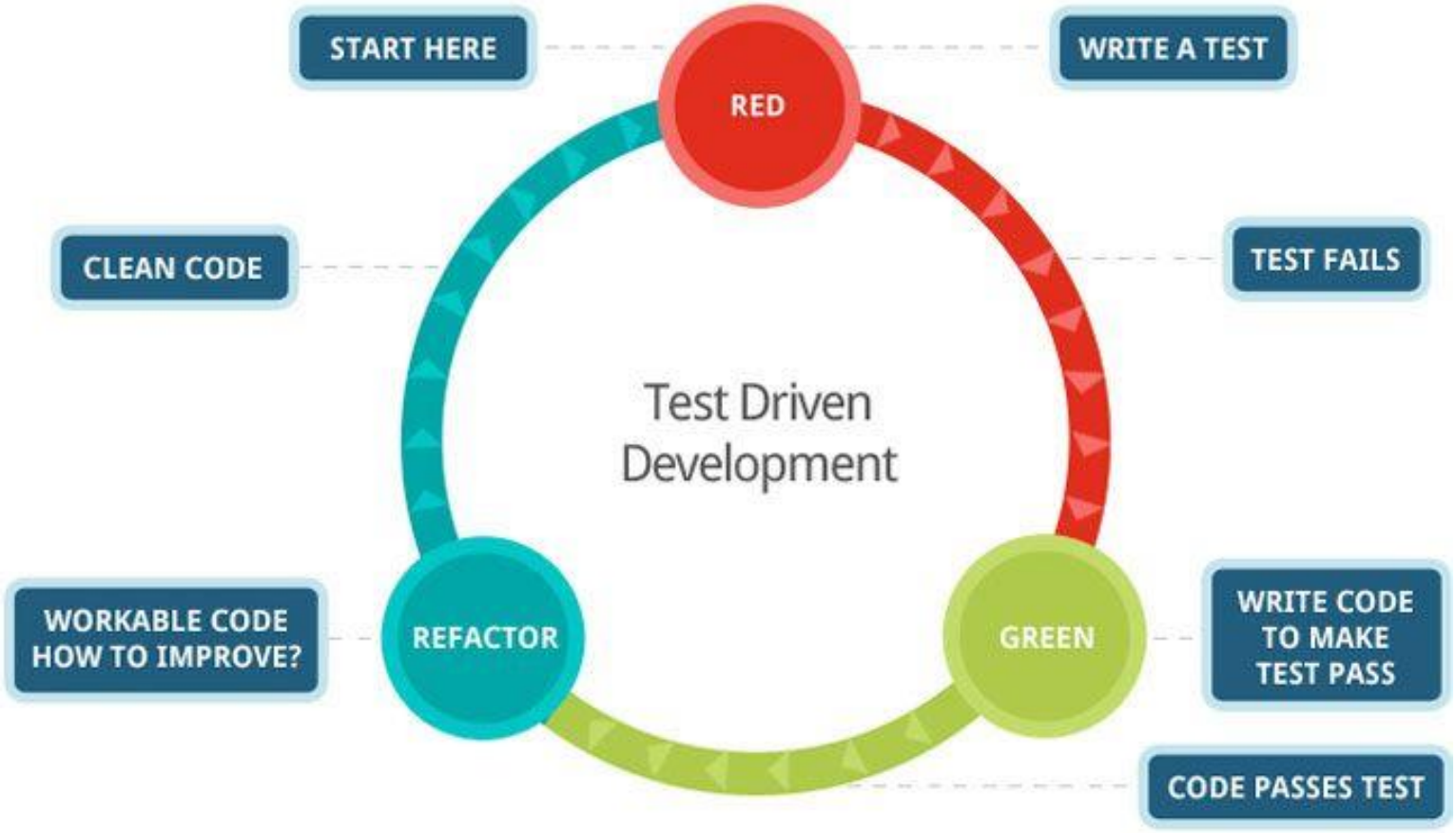
- Różne rodzaje testów są potrzebne.
- Testy powinny się uzupełniać.
- Mimo że warstwy wyglądają jakby się nakładały każda z nich testuje na innym poziomie abstrakcji

KLDiscovery

Podział Testów - niefunkcjonalne

- Security Testing
- Load Testing
- Performance Testing
- Accesibility Testing
- Usability Testing

Test Driven Development



KLDiscovery