Co dobry developer o testowaniu wiedzieć powinien.

Michał Szymczak

Będzie o...

- Tym, czy to developer ma testować
- Rodzajach testów
- Testach jednostkowych
- Zasadach pisania testów jednostkowych
- O testowalnym (i nie) kodzie
- O PdDD (TDD)

- Materiały dodatkowe
- Sesja Q&A

Część 1/8
[>_____]

Czy developer ma testować?

Czy developer ma testować?

Nie testujemy ponieważ...

- Nie wiadomo jak testować
- Testy długo się pisze a terminy gonią
- Testy wolno działają
- Klient placi za aplikację a nie za testy
- … ?
- Jesteśmy developerami, nie testerami

Czy developer ma testować?

Silo mentality

“A mind-set present in some companies when certain departments or sectors do not wish to share information with others in the same company. This type of mentality will reduce the efficiency of the overall operation, reduce morale, and may contribute to the demise of a productive company culture.”

Business dictionary (http://www.businessdictionary.com/definition/silo-mentality.html)

Czy developer ma testować?

Testowanie swojego kodu to...

- Błyskawiczny feedback
- Poczucie odpowiedzialności
  - błąd naprawia ten, kto popełnił błąd
- Mniejsza obiektywność
  - pisząc testy po napisaniu funkcjonalnośc powielamy schemat myślenia (golden path)
  - dlatego zalecane TDD

Czy developer ma testować?

Tak, developer powinien testować.

Jakie są rodzaje testów?

Jakie są rodzaje testów?

Rodzaje testów:

- Testy jednostkowe
  - Najszybsze, największe wymagania co do architektury aplikacji
- Testy integracyjne
- Testy akceptacyjne
  - Świadomość celu
- Testy manualne
  - Najwolniejsze, praktycznie brak wymagań co do architektury aplikacji

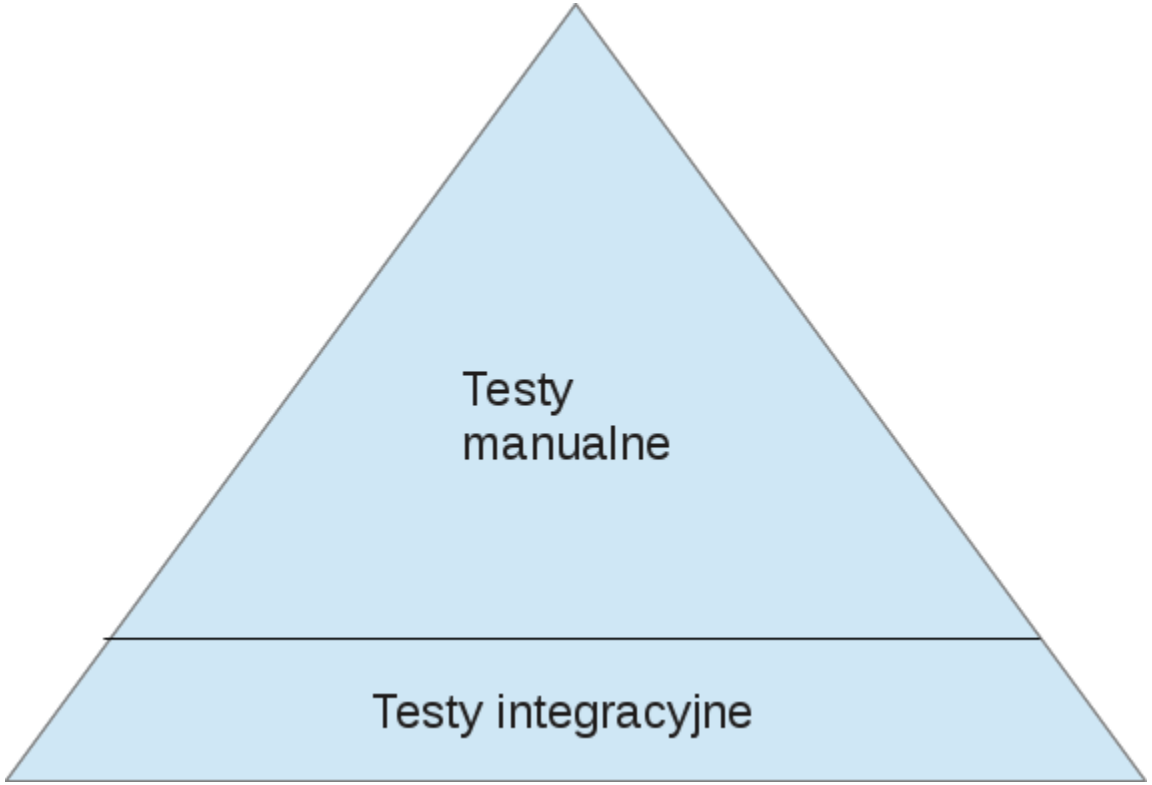
Jakie są rodzaje testów?

- Niestety, o decyzji jakie testy przeprowadzać często decyduje nie największa przydatność, a ograniczenia architektury (patrz: testowalny kod)
- Testy manualne vs automatyczne
  głupie lenistwo vs mądre lenistwo

Jakie są rodzaje testów?

Organizacja bez kultury testowania, w której ktoś kazał testować.

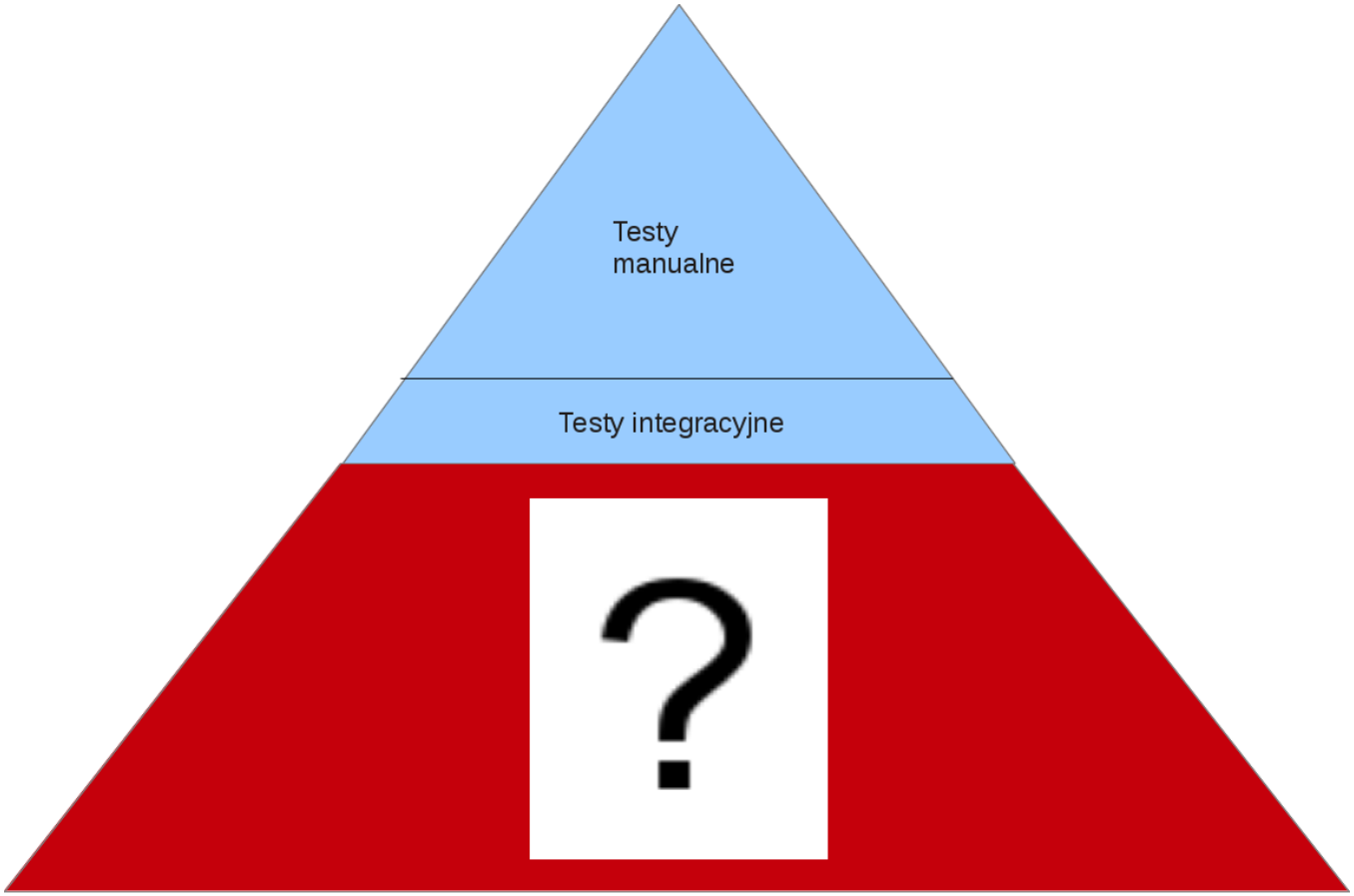
Rzekomo:



Jakie są rodzaje testów?

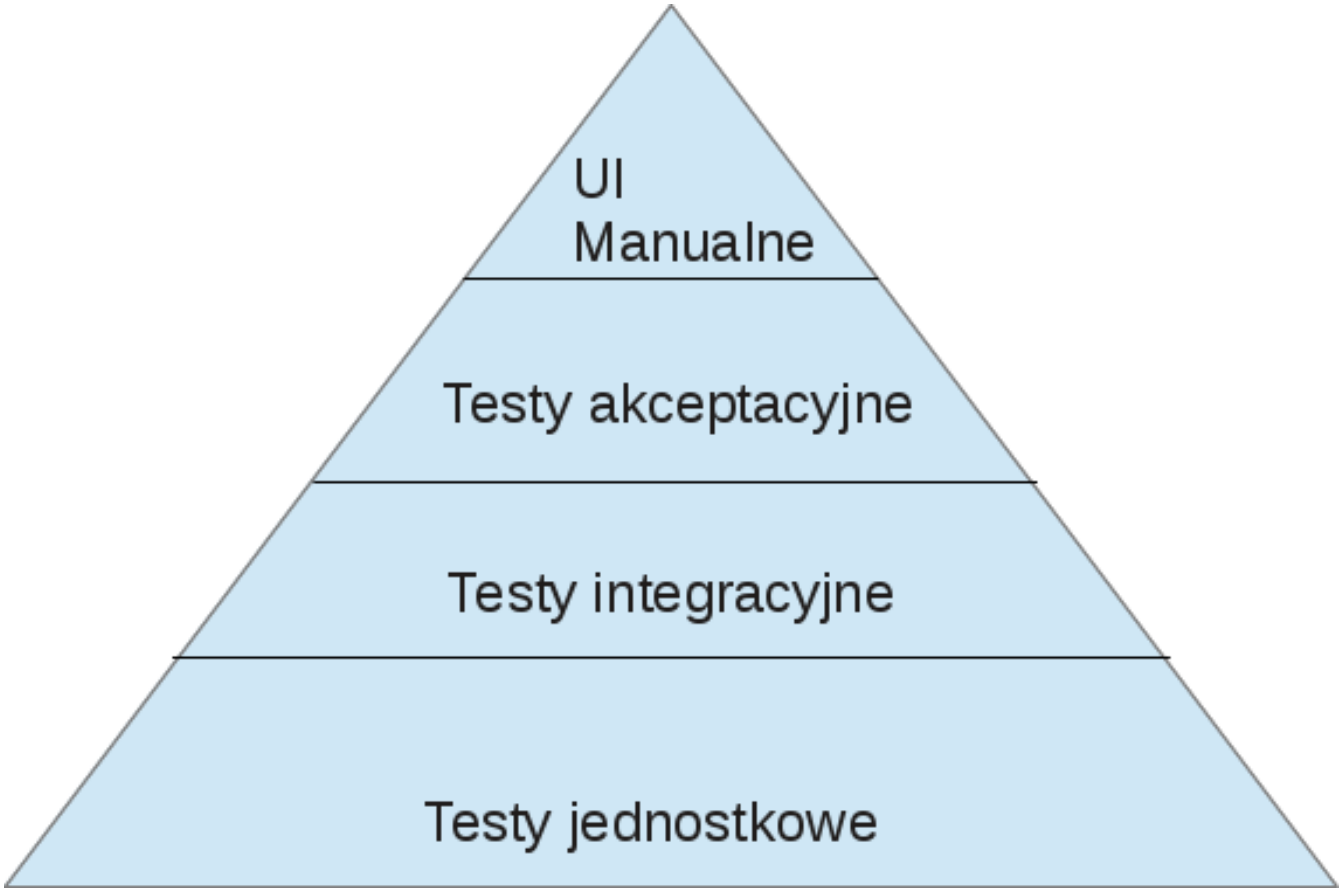
Organizacja bez kultury testowania, w której ktoś kazał testować.

W rzeczywistosci:



Jakie są rodzaje testów?

Organizacja z dojrzałą kulturą testowania:



Testy jednostkowe

Testy jednostkowe

Czym jest test jednostkowy

- **Test** weryfikujący poprawność działania **pojedynczego** elementu (jednostki) programu - np. **metody lub obiektu** w programowaniu obiektowym lub **procedury** w programowaniu proceduralnym.

[BankAccountTest] (Przykład testu jednostkowego)

Testy jednostkowe

Test nie jest testem jednostkowym jeśli:

- Komunikuje się z bazą danych
- Łączy się przez sieć
- Wykorzystuje system plików
- Nie może być uruchomiony razem z dowolnym innym testem jednostkowym
- Musisz wykonać specjalne czynności ze swoim środowiskiem aby go uruchomić (np. edytować plik konfiguracyjny)

[Michael Feathers, http://www.artima.com/weblogs/viewpost.jsp?thread=126923]

- Posiada zależności na które nie mamy wpływu podczas testowania i które mogą wpłynąć na wynik testu

[Michał Szymczak]

Testy jednostkowe

Korzyści ze stosowania testów jednostkowych

- dokładnie **wiemy gdzie** jest **problem**
- dzięki uniknięciu zależności, nie możemy ‘zwalić winy’ na inny obiekt
- wymusza **modularną architekturę**
- są bardzo **szybkie** (ułamki sekund na test)
- **wiarygodny** test **coverage** (@covers)
- **łatwiejsze TDD** - szybki feedback

Testy jednostkowe

Ograniczenia testów jednostkowych

- testują tylko **obiekty w izolacji** - nie testują interakcji pomiędzy obiektami (osobno mogą działać poprawnie, ale nie ‘dogadują’ się ze sobą)
- trudne do napisania gdy chcemy pokryć **legacy code** pisany przez ludzi, którzy nie mieli pojęcia, co dobry developer o testowaniu wiedzieć powinien

Zasady przy pisaniu testów jednostkowych

Zasady przy pisaniu testów jednostkowych

Nie powielaj logiki którą testujesz

- nie wyliczaj tylko daj gotową asercję
- w testach nie ma 'logiki', logika = bugi

[test02]

Zasady przy pisaniu testów jednostkowych

Nie usuwaj poprzednich testów

- nić Ariadny
- poprzednie testy przeciwdziałają regresji

[test03]

Zasady przy pisaniu testów jednostkowych

Testy to też kod

- stosuj dobre praktyki: KISS, DRY etc.
- rafaktoryzuj

[test04]

Zasady przy pisaniu testów jednostkowych

Test ma być wymowny

- jest to dokumentacja API
- nazwy metod dłuższe, opisowe
- metoda może opisywać warunki początkowe i rezultat

[test03 - tylko nazwy metod, sam kod potem]

Zasady przy pisaniu testów jednostkowych

Testy muszą być pewne

- przechodzą w 100%, jeśli nie - popraw najpierw kod/testy
- nie może być sytuacji w której testy nie przechodzą innej, niż bug w kodzie

[test05]

Zasady przy pisaniu testów jednostkowych

Testy sprawdzają jedną rzecz na raz

- ponieważ jeśli więcej, może nie być jasne co tak naprawdę się popsuło
- ponieważ są dokumentacją

[test06 + test06b]

Zasady przy pisaniu testów jednostkowych

Testy są niezależne

- Mogą być uruchamiane w dowolnej kolejności zawsze dając ten sam rezultat
- nie muszą 'sprzątać' po sobie (tearDown = prawdopodobnie to nie test jednostkowy)

Zasady przy pisaniu testów jednostkowych

Testuj pod kątem interfejsu, nie implementacji.

- testuj tylko publiczne metody, to ułatwia refaktoryzację
- testowanie prywatnych/chronionych metod zaciemnia dokumentację

Zasady przy pisaniu testów jednostkowych

Używaj zależności, którym ufasz w 100%

- stosuj stuby i mocki

Testowalny kod

Testowalny kod

Kodu nie da się testować jednostkowo gdy:

- pomieszany **operator new** z logiką aplikacji [code01]
  - nie da się stubować obiektów - nieuniknione zależności
  - jedyna dostępny układ to ten produkcyjny

- **metody statyczne w zależnościach**
  - nie da się w pełni stubować obiektu z metodami statycznymi gdy jest on zależnością

Testowalny kod

Kodu jest trudny do testowania jednostkowego gdy:

- napisany przez osoby niekompetentne, nie mające pojęcia o testowaniu
- architektura nie sprzyja
  - są zagnieżdżone instrukcje warunkowe - wiele ścieżek do przejścia, niektóre trudne do odtworzenia - zamiast tego lepiej stosować polimorfizm
  - brak zasady jednej odpowiedzialności
  - ogólny bałagan
  - pozorna obiektowość

PdDD / TDD

(Pomyśl dobrze Driven Development)

Test Driven Development

- kierunek ustalasz zanim wyruszysz w trasę, czy po 30km?
- “ if you need to explain to a computer how to check the requirement, you’ll need to be damn sure understand it yourself. If you don’t (and you often don’t) it’s much cheaper to find that out before you write the code.”

[Matt Wynne, http://blog.mattwynne.net/2012/11/20/tdd-vs-bdd/]

Test Driven Development

Możliwe sytuacje

- new feature
- free time
- bug found
- new feature conflict

Test Driven Development

New feature:

1. wymaganie
2. test
3. najprostszy kod
4. refaktoryzacja przy zielonym

Test Driven Development

Free time:

- refaktoryzacja testow przy zielonym
- refaktoryzacja kodu przy zielonym

Test Driven Development

Bug found:

1. analiza wymagania ktore nie jest spelnione gdy jest ten bug
2. test nieprzechodzący z powodu buga
3. naprawa buga
4. refaktoryzacja przy zielonym

Test Driven Development

New feature conflict:

1. zrozumienie testu ktory przestal przechodzic
2. porownanie z nowonapisanym testem
3. konsultacja z osoba decyzyjna
4. wyrzucenie/poprawienie testu ktory jest nieaktualny

Materiały obowiązkowe

- The art of Unit Testing
  - http://artofunittesting.com/
  - http://www.amazon.com/Art-Unit-Testing-Examples-Net/dp/1933988274
- The Clean Code Talks - Unit Testing
  - http://www.youtube.com/watch?v=wEhu57pih5w

Q & A

Q & A

Q: Nie wiadomo jak testować

A: Nauczyć, (np. wysłać na taki wykład, zrobić warsztaty etc.)

Q & A

Q: Testy długo się pisze a terminy gonią

A: Polityka firmy (Twoja jeśli freelancer) nie zachęca do pisania testów. Wytłumaczyć zarządowi/sobie co to są rosnące koszty utrzymania aplikacji oraz dług techniczny. Jeśli nadal nie rozumie czynność powtórzyć. Do skutku.

Q & A

Q: Testy wolno działają

A: Prawdopodobnie stosunek szybkich testów jednostkowych do wolnych testów niejednostkowych jest niekorzystny. Architektura może nie pozwalac pisac testow jednostkowych. Warto wprowadzić TDD/BDD aby nauczyć developerów myśleć o tym czy da się kod przetestować zanim napiszą jakąś głupotę. Podczas tworzenia aplikacji webowych warto zaszczepić zasadę Skinny Controller, Fat Model.

Q & A

Q: Klient placi za aplikację a nie za testy

A: Tak samo kierowcy myslą, że płacą za asfalt po którym jeżdżą samochody a nie 5 warstw pod nim. Że można przejechać po jezdni przez most a nie za betonowe fundamenty wpuszczane w dno rzeki. Co nie znaczy że można oszczędzać na bezpieczeństwie.

Q & A

Q: ...

A: ...

Co dobry developer o testowaniu wiedzieć powinien.

Michał Szymczak