



S01: Java is the new Black

E05 : Wprowadzenie do testowania na przykładzie JUnit5

Tomek
Owczarek
15/04/2019

Testowanie oprogramowania?

Czyli właściwie co?



Testy funkcjonalne

- Unit testing
- Component testing
- Integration testing
- System testing
- Sanity testing
- Smoke testing
- Interface testing
- Regression testing
- Beta/Acceptance testing
- Mutation testing
- ...

Testy niefunkcjonalne

- Performance Testing
- Load testing
- Stress testing
- Volume testing
- Security testing
- Compatibility testing
- User testing
- Browser Compatibility Testing
- Vulnerability Testing
- Install testing
- Recovery testing
- Reliability testing
- Usability testing
- Compliance testing
- Localization testing
- Accessibility testing
- Gorilla testing
- Monkey testing
- ...



Testy funkcjonalne

- ✓ **Unit testing**
- ✓ **Component testing**
- Integration testing
- System testing
- Sanity testing
- Smoke testing
- Interface testing
- Regression testing
- Beta/Acceptance testing
- Mutation testing
- ...

Testy niefunkcjonalne

- Performance Testing
- Load testing
- Stress testing
- Volume testing
- Security testing
- Compatibility testing
- User testing
- Browser Compatibility Testing
- Vulnerability Testing
- Install testing
- Recovery testing
- Reliability testing
- Usability testing
- Compliance testing
- Localization testing
- Accessibility testing
- Gorilla testing
- Monkey testing
- ...

Plan na dziś

1. Po co testy?
2. Dobre testy czyli jakie?
3. Jak testy jednostkowe mają się do innych typów?
4. Co testować?
5. Pogoda na testy czy testy na pogodę?

Tomek Owczarek



Expert Software Engineer @ **TomTom** 

Java Trainer @



software
development
academy



<https://github.com/towczare>



<https://www.linkedin.com/in/towczarek/>



Weryfikacja

(nie ufaj nawet swojemu kodowi)

Jakość

(nie jakoś)

Dokumentacja

(najlepsza, najbardziej aktualna)

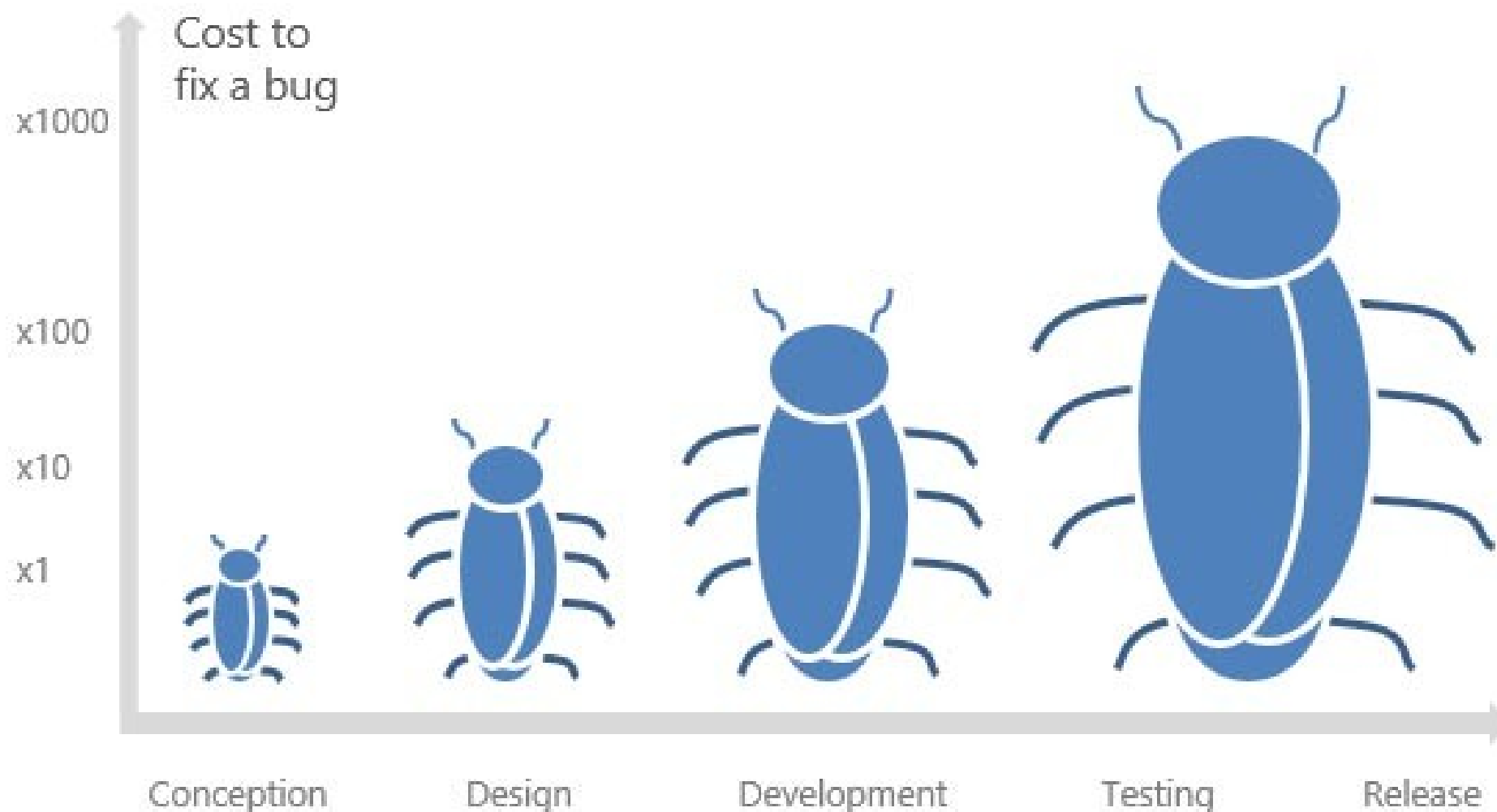
Spryt

(szybsza pętla zwrotna)

Reasumując - WJDS czyli

- 1. Weryfikacja** - wyłapywanie i unikanie błędów podczas fazy tworzenia oprogramowania, zarówno od strony technicznej jak i koncepcyjnej.
- 2. Jakość** - tworzenie rozwiązań lepszej jakości, które łatwiej utrzymywać, opartych na lepszym designie.
- 3. Dokumentacja** - najlepsza możliwa dokumentacja.
- 4. Spryt** - szybszy feedback, programowanie z większą pewnością

Koszt wykrycia błędów względem czasu





JAVA **FAKTURA**

Kupuj drogo, sprzedawaj tanio, powtórz czyli historia upadku Knight Capital Group



*Third, there had been substantial code refactorings in SMARS over the years **without thorough regression testing;***

Współczesny Ikar czyli lot Ariane 5 501



*While there was some unit testing and integration testing with A4 data, neither end-to-end, integration testing with hardware and software **nor test simulations with realistic data from the A5 trajectory data were ever performed.***

Winter is coming czyli termostaty Nest





JAVA  **FAKTURA**

Łatwo definiowalne

(testów nie powinno się ciężko pisać)

Konkretne

(test jednostkowy powinien sprawdzać jedną konkretną rzecz)

Szybkie

(unikaj IO, DB i innych zależności spowalniających)

Niezbyt liczne

(testuj tylko API, unikaj testowania szczegółów implementacyjnych)

Reasumując

- 1. Łatwo definiowalne**
- 2. Konkretne**
- 3. Szybkie**
- 4. Niezbyt liczne**



Zagadka

Co mają wspólnego?

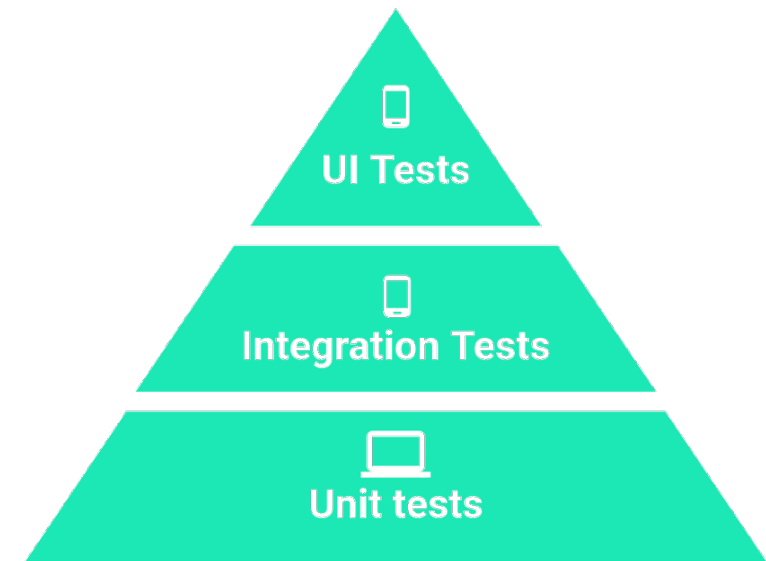


```
flavio@Flavios-MacBook-Pro: ~/dev/jest
# flavio @ Flavios-MacBook-Pro in ~/dev/jest [17:47:58]
$ yarn test
yarn test v0.27.5
$ jest
PASS ./math.test.js
  ✓ Adding 1 + 1 equals 2 (6ms)
  ✓ Multiplying 1 * 1 equals 1
  ✓ Subtracting 1 - 1 equals 0 (1ms)
  ✓ Dividing 1 / 1 equals 1 (1ms)

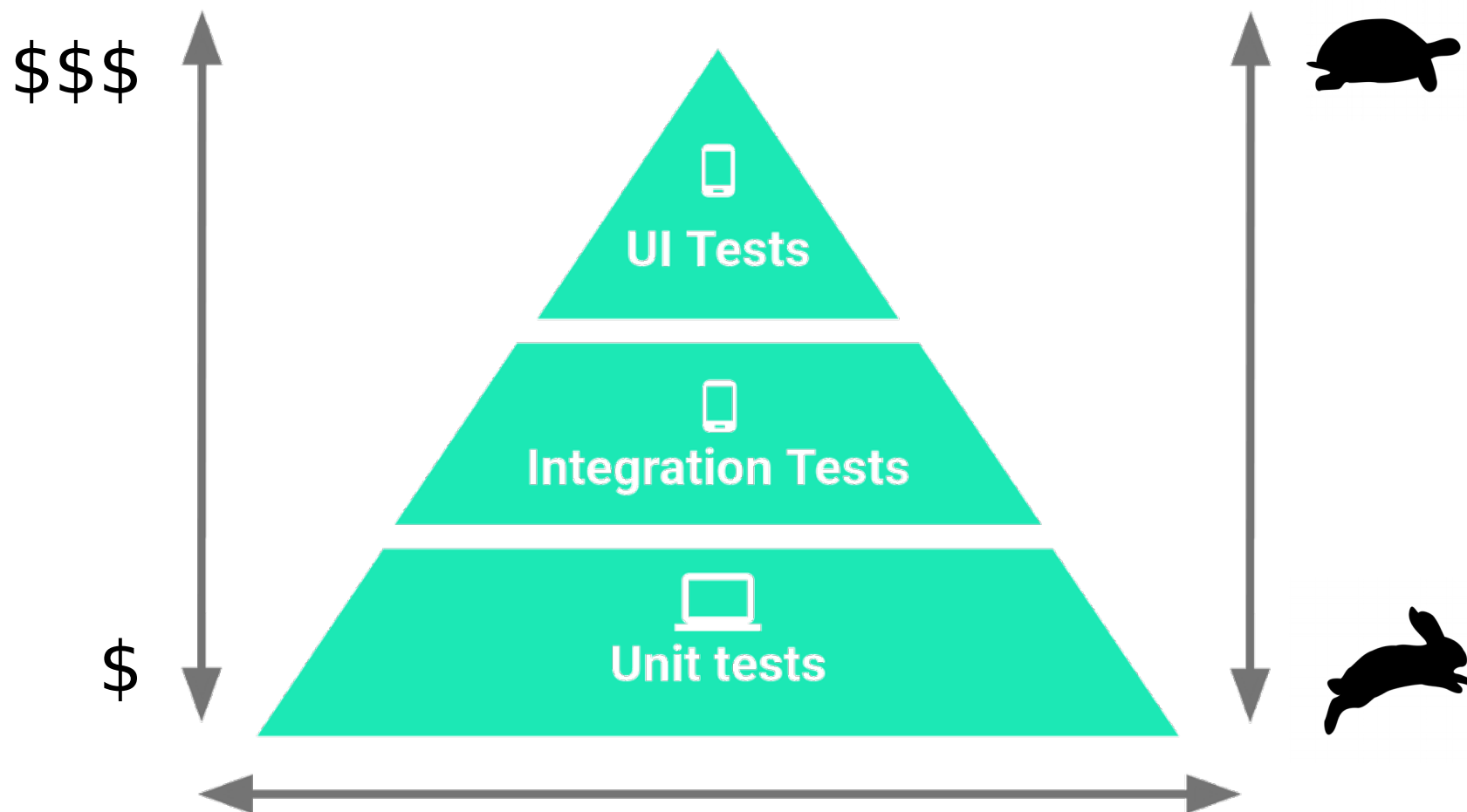
Test Suites: 1 passed, 1 total
Tests: 4 passed, 4 total
Snapshots: 0 total
Time: 2.178s
Ran all test suites.
Done in 3.54s.

# flavio @ Flavios-MacBook-Pro in ~/dev/jest [17:51:25]
$
```



Piramidy



Piramida testów



Priorytety testowania

- 
- 1. Kod którego się boimy**
 - 2. Daleki logiczny fragment w kodzie**
 - 3. Bug przed poprawkami**
 - 4. Instrukcje warunkowe if/for/while ...**
 - 5. Wyjątki**
 - 6. Trywialne**
 - 7. Legacy code**

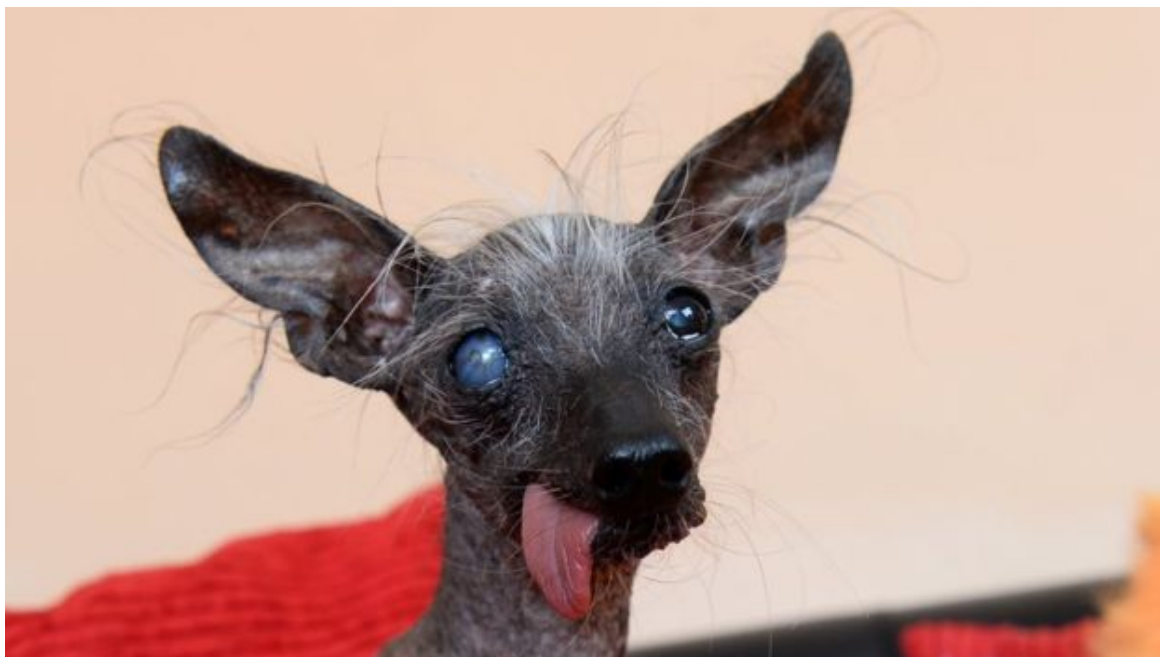
CHCEMY KODU!

REGUŁY

<https://github.com/towczare/s01e05>

REGUŁY

<https://github.com/towczare/s01e05>



Brzydkie przykłady

REGUŁY

<https://github.com/towczare/s01e05>



Brzydkie przykłady



Dobre przykłady

TESTY WITAJCIE!

<https://github.com/towczare/s01e05/tree/master/01>

Co warto zapamiętać?

1. Używaj **literałów** w swoich testach
2. Dbaj o **dobre nazwy** swoich testów
3. Konwencja **given** / **when** / **then** pozwala na uporządkowanie testów

Pięćdziesiąt twarzy smogu!

<https://github.com/towczare/s01e05/tree/master/02>

Co warto zapamiętać?

1. **Parametryzowane testy** to świetny sposób na pozbycie się kodu boiler-plate
2. **DDT** - czyli testy sterowane danymi
3. Konkretnie testy!
4. Możliwość importu zewnętrznych danych w postaci csv
5. Polska to wbrew opinii bardzo tolerancyjny kraj (przynajmniej w stosunku do smogu)

Tylko dobra pogoda (bez) wyjątków!

<https://github.com/towczare/s01e05/tree/master/03>

Co warto zapamiętać?

1. **API bibliotek** do testowania z reguły przewidziało większość twoich wymagań (bez wyjątku dla testowania wyjątków)
2. Tworzenie **nowych klas wyjątków** nie zawsze jest dobrym pomysłem
3. Wprowadzaj też kontrolne testy wykraczające poza tzw. **happy path** (co jeśli omawiany przykład, zawsze rzucał wyjątkiem? ;-))

Przeminęło z wiatrem! (czy anemik poleci)

<https://github.com/towczare/s01e05/tree/master/04>

Co warto zapamiętać?

1. Nie testuj **prywatnych** metod
2. Testuj **API** - nie testuj implementacji
3. Poluj na **warunki brzegowe**, nie unikaj ich
4. **DDD** - klasy domenowe mogą być dużo przyjemniejsze do testowania w przeciwieństwie do niektórych serwisów
5. Myślenie **obiektywne** upraszcza kod i sprawia, że jest on również łatwiejszy w testowaniu (np. przez większą separację)

Gdzie puścić latawiec? (podejście TDD)

<https://github.com/towczare/s01e05/tree/master/05>

Co warto zapamiętać?

1. **Test** - **Test** - **Refaktor** - ↺
2. Nie musisz trzymać się kurczowo wszystkich faz
3. Testy w **TDD** są tylko narzędziem w metodyce rozwijania oprogramowania, nie celem samym w sobie
4. Nie wszystkie problemy powinniśmy rozwiązywać przy użyciu TDD, ale dla wielu nietrywialnych mogą sporo ułatwić
5. Programowanie jest fajniejsze od puszczania latawców (i można je uprawiać nawet podczas deszczu)

Dziękuję za uwagę

JAVA  FAKTURA

