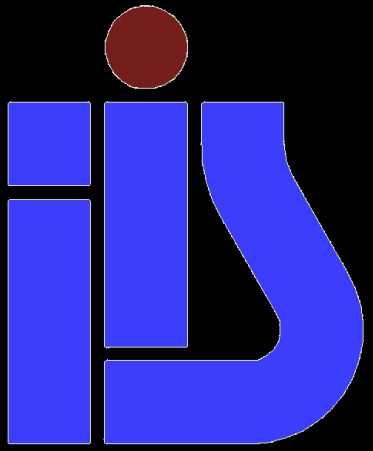


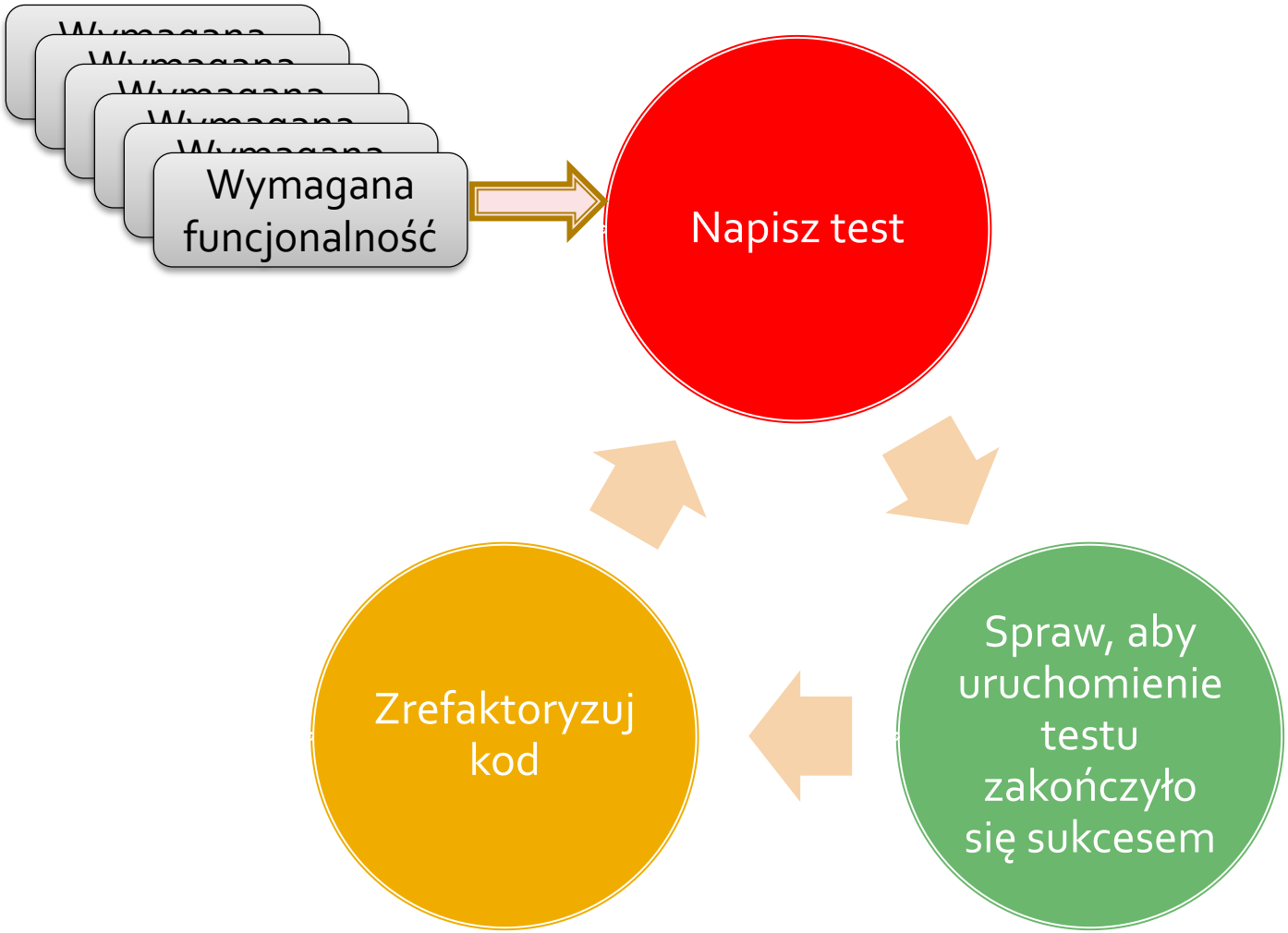
W formie kata...

Test Driven Development

Co to jest Test Driven Development (TDD)

- Iteracyjny proces wytwarzania oprogramowania w którym:
  - Testy jednostkowe przesunięte są „do przodu”
  - Iteracja rozpoczyna się od napisania testu
    - Który staje się częścią specyfikacji tego co mamy zaimplementować
  - Iteracje są krótkie (minuty)
    - Szybka pętla zwrotna
  - Pokrycie kodu testem jest bardzo wysokie

Cykl TDD


Wymagana funcjonalność
Napisz test
Spraw, aby uruchomienie testu zakończyło się sukcesem
Zrefaktoryzuj kod

3 prawa (zakazy) TDD

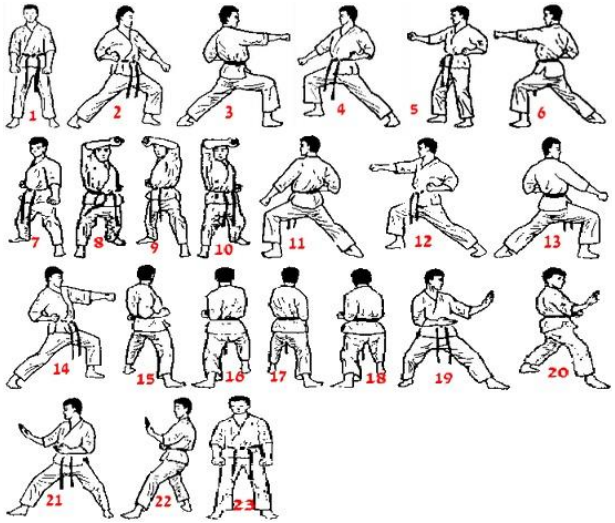
1. Nie napiszesz żadnego kawałka kodu produkcyjnego bez uprzedniego napisania testu jednostkowego, którego wykonanie zakończy się błędem.
2. Nie napiszesz więcej kodu testu jednostkowego niż jest to potrzebne do jego wykonania zakończonego błędem.
3. Nie napiszesz więcej kodu produkcyjnego niż jest to wystarczające do poprawnego wykonania testu jednostkowego.

Po co te reguły?

- Kiedy zaczniemy je łamać:
  - Rozpraszamy się
  - Sprawy się komplikują
  - Zaczynamy zapominać o niektórych krokach
  - Kończymy mając nadzieję, że undo w naszym edytorze ma wystarczający bufor ☺

Kata

- Wysoce sformalizowany rodzaj ćwiczeń stosowanych w wielu tradycyjnych sztukach i sportach walki, szczególnie z grupy budō, jak również taekwondo Wikipedia
- Kata dotyczy:
  - formy,
  - dążenia do perfekcji



Coding Kata

- Ćwiczenie programistyczne wykonywane w celu doskonalenia umiejętności poprzez praktykę i systematyczne powtarzanie
  - Wykorzystamy je jako metodę wprowadzenia do TDD
  - Ale można je wykorzystać do:
    - Doskonalenia w programowaniu,
    - Poznawania nowych języków, platform, narzędzi,
    - Rozgrzewki przed prawdziwym zadaniem,
    - …

Coding Dojo

- Spotkania w ramach których kata wykonuje się w grupie.
  - Dzielenie się własnym doświadczeniem
  - Podpatrywanie sposobów w jaki pracują inni
  - Metoda szybkiego uczenia się programowania

Temat na zajęcia

- Napisz prostą aplikację, która umożliwia rozlokowanie wirtualnych maszyn pomiędzy serwery.
- Rozwiązanie musi dystrybuować maszyn wirtualne w najbardziej zrównoważony sposób.
- Jako parametry otrzymujemy liczbę dostępnych serwerów, oraz liczbę maszyn wirtualnych do rozmieszczenia.
- Należy poradzić sobie z faktem, że:
  - Każdy serwer może udostępniać różną przestrzeń możliwą do wykorzystania przez maszyny wirtualne (pojemność serwera).
  - Maszyny wirtualne mogą mieć różny rozmiar – zapotrzebowanie na przestrzeń serwera.

Plan

- Prezentacja „na żywo”
- Powtarzamy w parach
  - Jeden pisze test drugi implementuje, zmiana
- Indywidualne kata
- Dodatkowe pomysły na testy

Zaliczenie

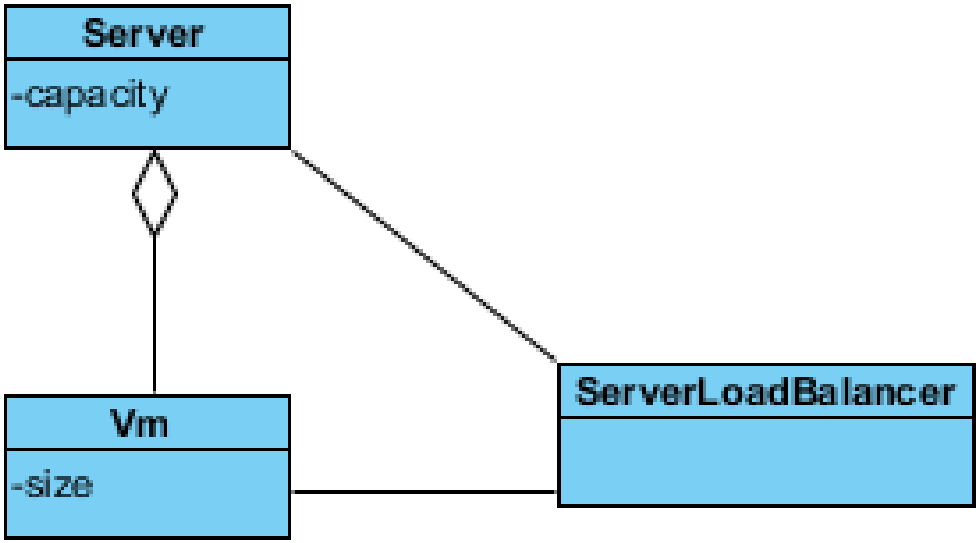
1. Nauczenie się naszego kata
2. Znalezienie i przećwiczenie innego
   - http://codingdojo.org/cgi-bin/index.pl?KataCatalogue
   - http://codekata.com/
   - http://katas.softwarecraftsmanship.org/

- Ad 1. Sprawdzenie indywidualne – na ostatnich zajęciach
- Ad 2. wykazanie się poprzez repozytorium z zapisem ćwiczenia

Testy do zaimplementowania

1. Przypadek bazowy – brak maszyn do uruchomienia
2. Jeden serwer i jedna maszyna zajmująca cały serwer
3. Jeden serwer i jedna maszyna zajmująca część zasobów serwera
4. Jeden serwer i kilka maszyn mieszczących się na serwerze
5. Dwa serwery jedna maszyna - przypisanie maszyny do mniej obciążonego serwera
6. Serwer zbyt obciążony na przyjęcie maszyny
7. Równomiernie rozmieszczenie kilku maszyn pomiędzy dostępnymi serwerami

Model



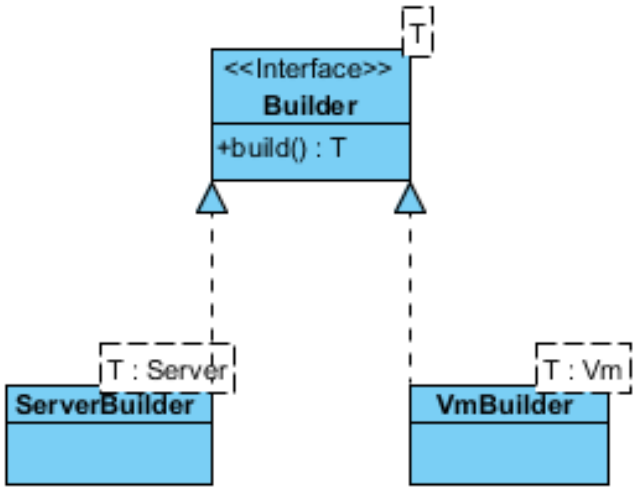
Test 1 - Przypadek bazowy – brak maszyn do uruchomienia

1. Pojedynczy serwer, brak maszyn, serwer pozostaje pusty
    1. balancingServer_noVm_ServerStaysEmpty
2. Budujemy podstawy struktury (głównie testu)
3. Matchery
4. Refaktoryzacja
    1. logika porównywania liczb zmiennoprzecinkowych
        1. Extract to local variable (replace all occurences), extract to method, inline local variable
    2. Factory method:
        1. server(),
        2. hasCurrentLoadPercentageOf(),
    3. Epsilon constant
4. Zapis video

Test 2 - Jeden serwer i jedna maszyna zajmująca cały serwer

1. balancingOneServerWithOneSlotCapacity_andOneSlotVm_fillsTheServerWithTheVm
2. Początek budowania logiki balancera
3. Refaktoryzacja
    1. Utworzenie interfejsu - Builder<T>
    2. Factory method - vm()
4. Zapis video

Wzorzec Builder wykorzystujący generics



Test 3 - Jeden serwer i jedna maszyna zajmująca tylko część zasobów serwera

1. balancingOneServerWithTenSlotsCapacity_andOneSlotVm_fillTheServerWithTenPercent
2. Test już nie powoduje błędów kompilacji
3. Wymusza zaimplementowanie logiki obliczania obciążenia serwera
4. Refaktoryzacja:
    1. Metoda addVm,
    2. stała zamiast magic numer
5. Zapis video

Test 4 - Jeden serwer i kilka maszyn mieszczących się na serwerze

1. Dodajemy obsługę więcej niż jednej maszyny (cały czas jeden serwer).
    1. balancingAServerWithEnoughRoom_getsFilledWithAllVms
    2. Wymusza modyfikację logiki metody balance oraz dodanie logiki pozwalającej na liczenie maszyn obsługiwanych przez serwer
2. Refaktoryzacja
    1. static hasVmCountOf
3. Zapis video

Test 5 - Dwa serwery jedna maszyna

1. przypisanie maszyny do mniej obciążonego serwera
    1. aVm_shouldBeBalanced_onLessLoadedServerFirst
2. Dodajemy drugi serwer
3. Obsługa logiki wyboru mniej obciążonego serwera do hostowania maszyny
    1. Trzeba mieć mechanizm do wstępnego „obciążania serwera”.
    2. Refaktoryzacja
        1. Upublicznienie stałej MAXIMUM_LOAD
        2. addInitialLoad(server)
        3. extractLessLoadedServer(Server[] servers)
4. Zapis video

Test 6 - Serwer zbyt obciążony na przyjęcie maszyny

1. Dodanie obsługi sytuacji w której rozmiar maszyny jest większy niż bieżąca pojemność serwera
   1. balanceAServerWithNotEnoughRoom_shouldNotBeFilledWithAVm
2. Refaktoryzacja
   1. findServersWithEnoughCapacity
   2. addToCapableLessLoadedServer
   3. DRY w klasie Server
3. Zapis video

Test 7 - Równomiernie rozmieszczenie kilku maszyn pomiędzy dostępnymi serwerami

1. Dodanie obsług wielu serwerów i wielu maszyn
    1. balance_serversAndVms
2. Refaktoryzacja
    1. Hermetyzacja currentLoadPercentage, capacity oraz size

3. Zapis video