# Inne wzorce architektoniczne

## Projektowanie i programowanie systemów internetowych II

mgr inż. Krzysztof Rewak

13 grudnia 2018

Wydział Nauk Technicznych i Ekonomicznych
Państwowa Wyższa Szkoła Zawodowa im. Witelona w Legnicy

# Plan prezentacji

1. MVC
2. IoC
3. ETL
4. CQS/CQRS
5. Podsumowanie

# MVC

# Czy MVC to remedium na wszystko?

Nie.

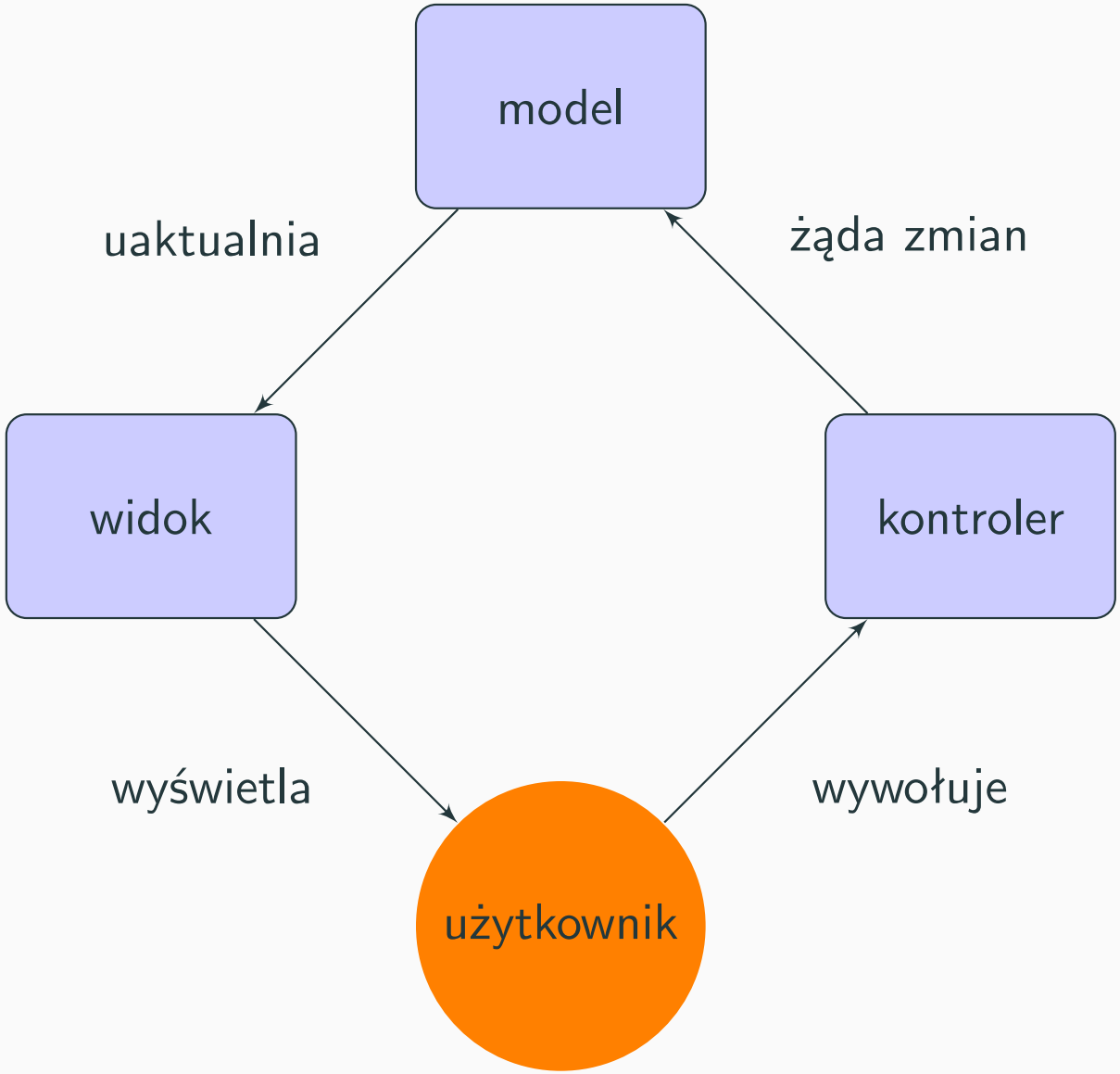
# Podsumowanie

# Czy MVC to remedium na wszystko?

MVC, nie dość, że bardzo często źle rozumiane, jest również bardzo często źle wykorzystywane.

Często też uważa się, że jest odpowiedzią na każdy architektoniczny problem.

# MVC


model
uaktualnia
żąda zmian
widok
kontroler
wyświetla
wywołuje
użytkownik

# MVC

**Model** odpowiada za... mapowanie danych? Niekoniecznie. Niektórzy autorzy wręcz mówią, że modelem jest cała warstwa domeny projektu.

# MVC

**Widok** to praktycznie rzecz biorąc wystawienie danych. Zazwyczaj myślimy o zbudowanym HTML-u, ale czy odpowiedź JSON też będzie widokiem?

# MVC

**Kontroler** natomiast powinien tylko i wyłącznie odebrać żądanie, przekazać je modelowi (w jakikolwiek sposób podzielonemu na warstwy!) i zwrócić znów użytkownikowi.

# MVC

Większość współczesnych frameworków webowych stosuje (albo nazywa inkorporuje do nazwy) wzorzec MVC. Czasami jednak trzeba sięgnąć po coś więcej, aby zbudować coś lepiej.

# IoC

# Odwrócenie sterowania

***Inversion of Control***, czyli *odwrócenie sterowania* polega na ”przeniesieniu funkcji sterowania wykonywaniem programu do używanego frameworku”.

# Dekonstrukcja definicji

*Klasyczne* podejście do programowania obiektowego polega na tworzeniu nowych obiektów i wywoływaniu na nich metod.

Programista kodując steruje zachowaniem programu.

# Dekonstrukcja definicji

IoC polega na tym, że dostarczony framework steruje programem, a programista jedynie podłącza się w odpowiednim momencie.

# Zasada Hollywood

*Nie dzwoń do nas. My oddzwonimy do ciebie.*

# Wstrzykiwanie zależności

Najpopularniejszą implementacją odwrócenia sterowania jest wstrzykiwanie zależności.

Należy pamiętać, że pojęcia te nie są tożsame.

# Wstrzykiwanie zależności

Przy okazji omawiania testowania przedstawiono przykład różnych implementacji testowania; która jest nalepsza?

```
public function send(): void {
    $mailer = new Mailer;
    $mailer->send();
}
```

czy może:

```
public function send(Mailer $mailer): void {
    $mailer->send();
}
```

czy może:

```
public function send(MailerInterface $mailer): void {
    $mailer->send();
}
```

# Wstrzykiwanie zależności

Dzięki IoC możemy uniezależnić wykonanie zadania od jego implementacji. Na poziomie wyższej warstwy zostanie stworzony obiekt implementujący odpowiedni interfejs i przekazany do kontrolera, serwisu lub innej wykonawczej klasy.

# Plusy?

- SOLID-nie porządkuje kod;
- zwiększa elastyczność naszego oprogramowania;
- zmniejsza możliwość wywołania efektów ubocznych czy modyfikacjach;
- ułatwia testowanie.

# Minusy?

- zwiększa skomplikowanie kodu?

# Życie po wstrzykiwaniu zależności

Co jeszcze jest implementacją IoC?

- wzorzec lokalizatora usług (*service locator*)
- operacyjny wzorzec metody szablonowej (*template method*)
- operacyjny wzorzec strategii (*strategy*)

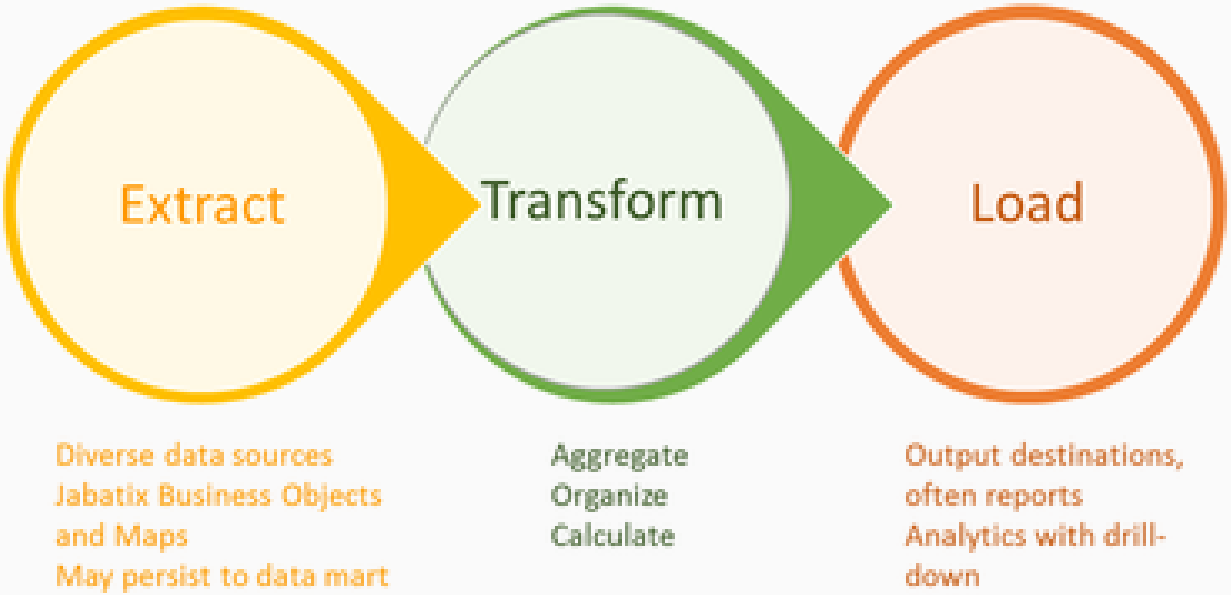
O wszystkich będzie mowa na przyszłosemestralnych zajęciach *Zaawansowane metody programowania*.

# ETL

# ETL

**ETL**, *extract*, *transform*, *load* (ang. *pobierz*, *zmień*, *wgraj*) to kolejny przydatny w biznesie wzorzec architektoniczny.

# ETL




**Rysunek 1:** `https://jabatixde.wordpress.com/2017/03/10/private-basic-pattern-high-value-extract-transform-load-with-jabatix/`

# *Extract*

**Extract** polega na przyjęciu w dowolny sposób danych wejściowych.

# *Transform*

**Transform** polega na ustandaryzowaniu danych wejściowych wedle wybranego schematu.

# *Load*

**Load** polega na przesłaniu ustandaryzowanych danych dalej.

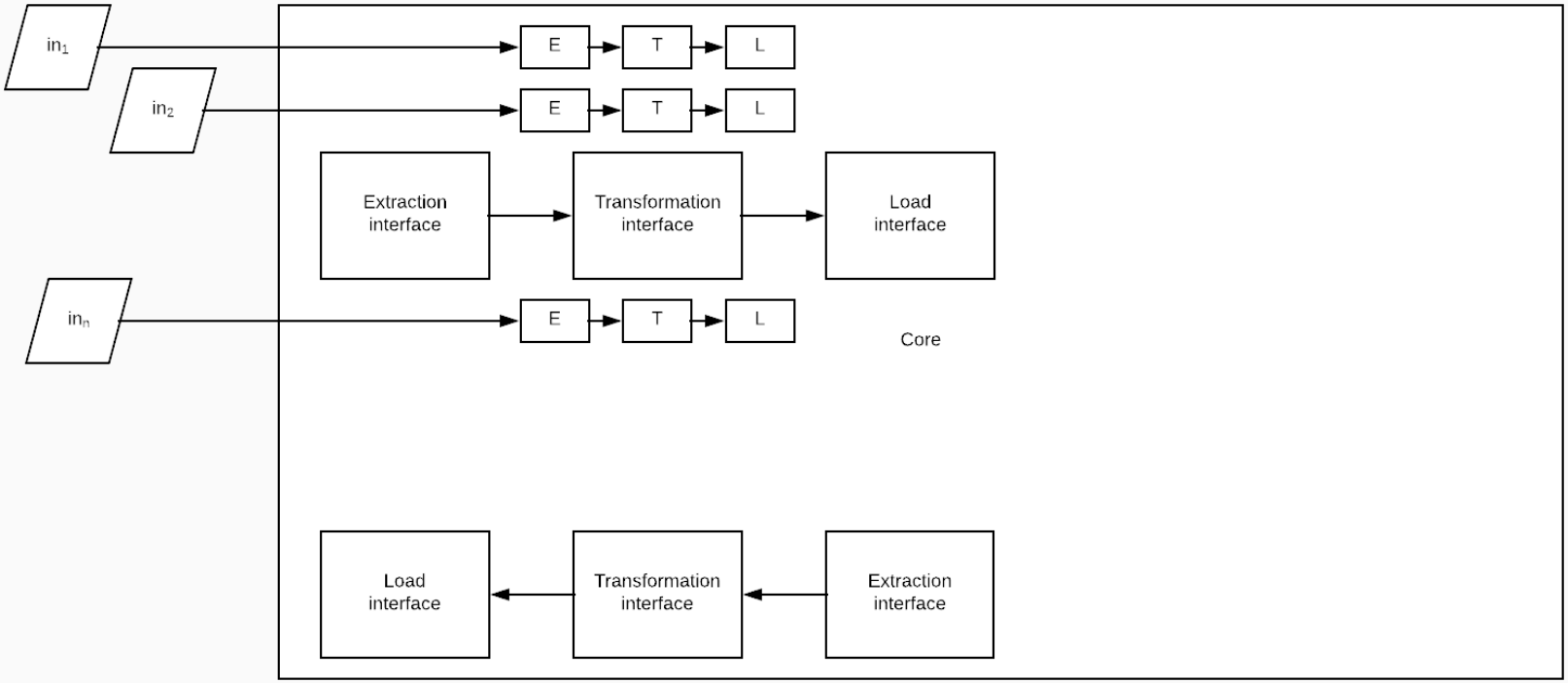
# ETL

**ETL** zatem może służyć jako system łączący inne systemy i dokładnie w tym celu jest często implementowany.

# ETL + IoC?

Ciekawym rozwiązaniem może być zastosowanie wzorca strategii przy ETL, żeby stworzyć system bramek API.

# ETL


in1
in2
inn
E
T
L
E
T
L
Extraction interface
Transformation interface
Load interface
E
T
L
Core
Load interface
Transformation interface
Extraction interface

# CQS/CQRS

# CQS

**CQS**, *Command Query Separation* (ang. *odseparowanie zapytań od poleceń*) to bardziej metoda programowania niż osobno zdefiniowany wzorzec architektoniczny, ale ostatnimi czasy stała się bardzo modna, więc warto pokrótce o niej wspomnieć.

# CQS

**Command** to metoda zmieniająca stan aplikacji, natomiast **Query** to metoda zwracająca stan aplikacji.

# CQS

*Pytanie nie powinno zmieniać odpowiedzi.*

# (nie)CQS

```
public Response create(string id) {
    User user = Users.create(id);
    return response(user);
}
```

# CQS

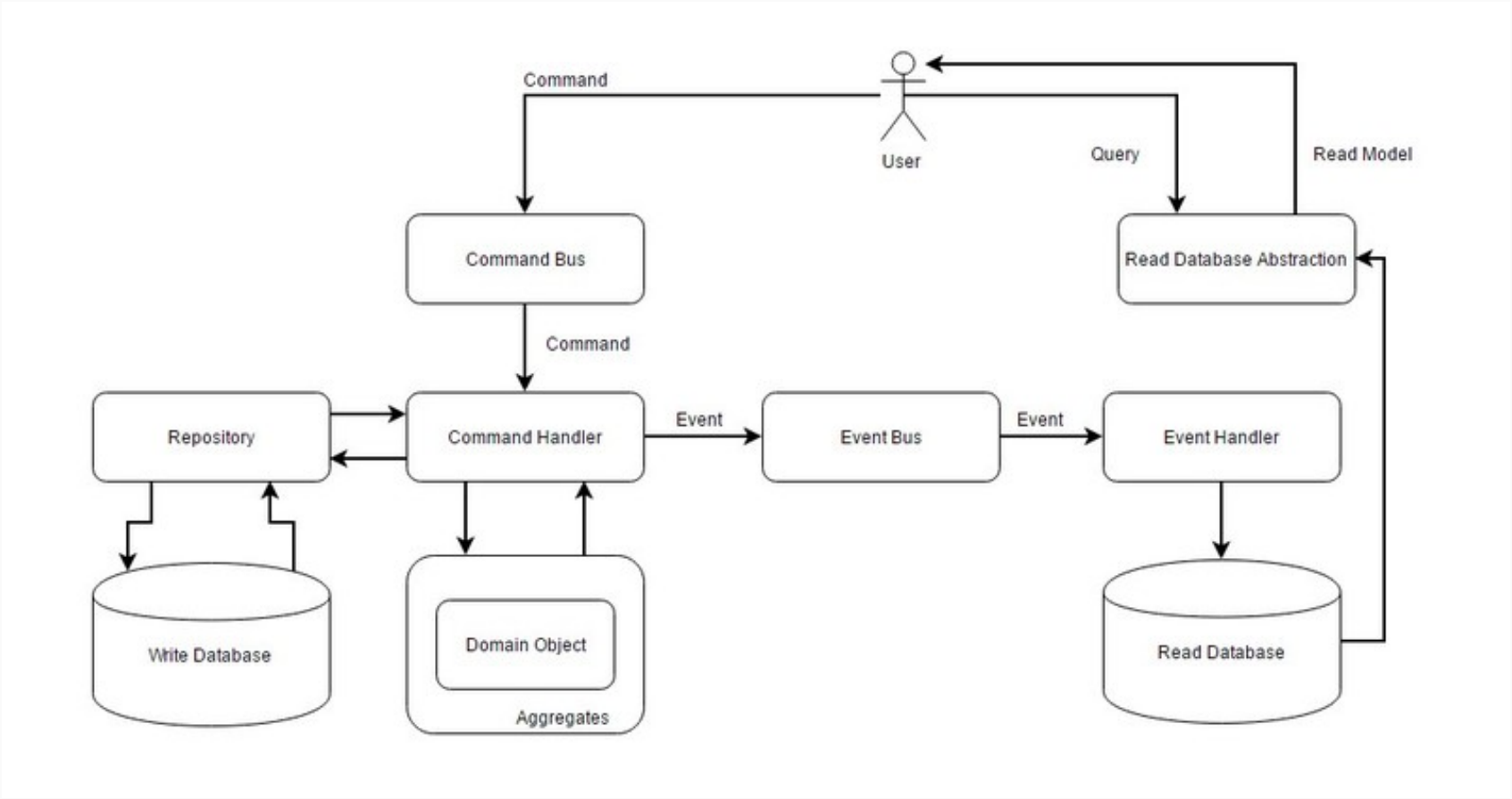
```
public void create(string id) {
    Users::create(id);
}

public Response get(string id) {
    return response(Users.find(id));
}
```

# CQRS

**CRS**, *Command Query Responsibility Segregation* (ang. *segregacja odpowiedzialności zapytań i poleceń*) to rozszerzenie CQS polegające na tworzeniu nie tylko metod, ale całych klas odpowiedzialnych za zapytania i polecenia.

# CQRS


Command
User
Query
Read Model
Command Bus
Read Database Abstraction
Command
Repository
Command Handler
Event
Event Bus
Event
Event Handler
Write Database
Domain Object
Aggregates
Read Database

# Podsumowanie

# Bibliografia i ciekawe źródła


- Gamma, Helm, Johnson, Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, 1994
- `http://commitandrun.pl/2016/05/30/Brutalne_prawdy_o_MVC/`
- `https://pl.wikipedia.org/wiki/Odwrócenie_sterowania`
- `https://www.sitepoint.com/inversion-of-control-the-hollywood-principle/`
- `https://bulldogjob.pl/articles/122-cqrs-i-event-sourcing-czyli-latwa-droga-do-skalowalnosci`

# Pytania?

Kod prezentacji dostępny jest w repozytorium git pod adresem
`https://bitbucket.org/krewak/pwsz-ppsi2`

Wszystkie informacje dot. kursu dostępne są pod adresem
`http://pwsz.rewak.pl/kursy/6`