Projekt

...

Piotr Jędrzejec

linkedin.com/in/piotr-jedrzejec

- ~2 lata w Sii
- Prawie 14 lat w zawodzie
- Głównie branża turystyczna
- Travelone.pl
- Fru.pl
- Sabre
- Ketle, squash, biohacking

Jak się podobało do tej pory ?

Rozkład jazdy - część 1

Slot 1:	9:00 - 10:20	Kawusia, wprowadzenie i zaczynamy od analizy wymagań
Slot 2:	10:30 - 11:40	Modelujemy zależności oraz start implementacji
Slot 3:	11:50 - 13:00	Kontynuacja implementacji (pure Java)
Przerwa:	13:00 - 14:00	
Slot 4:	14:00 - 15:30	Aktywacja Spring Boot'a
Slot 5:	15:40 - 17:00	REST API

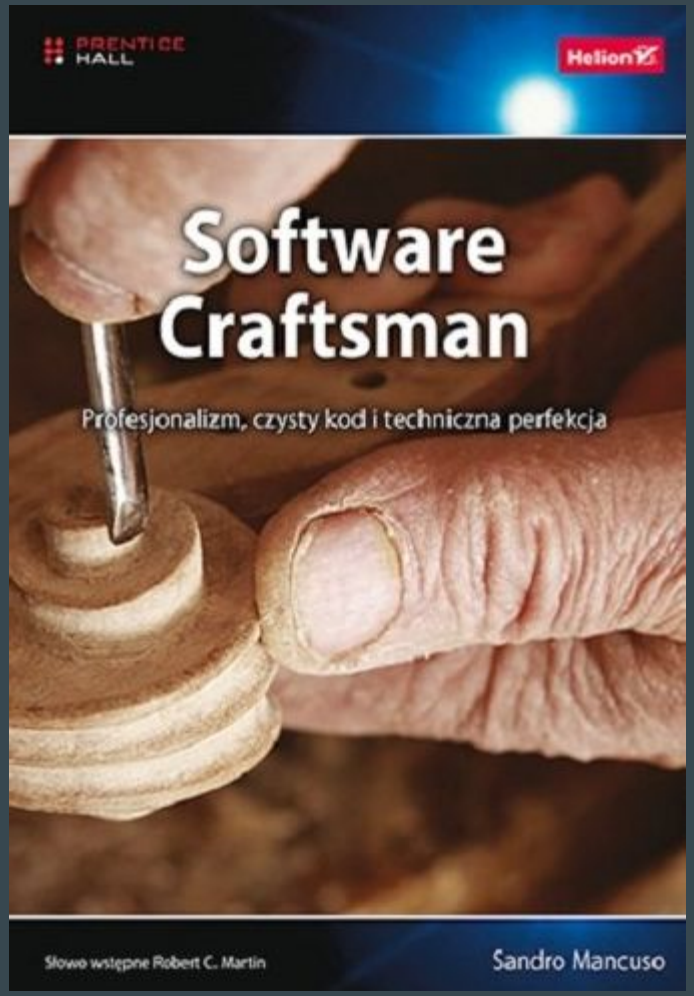
Slot 1

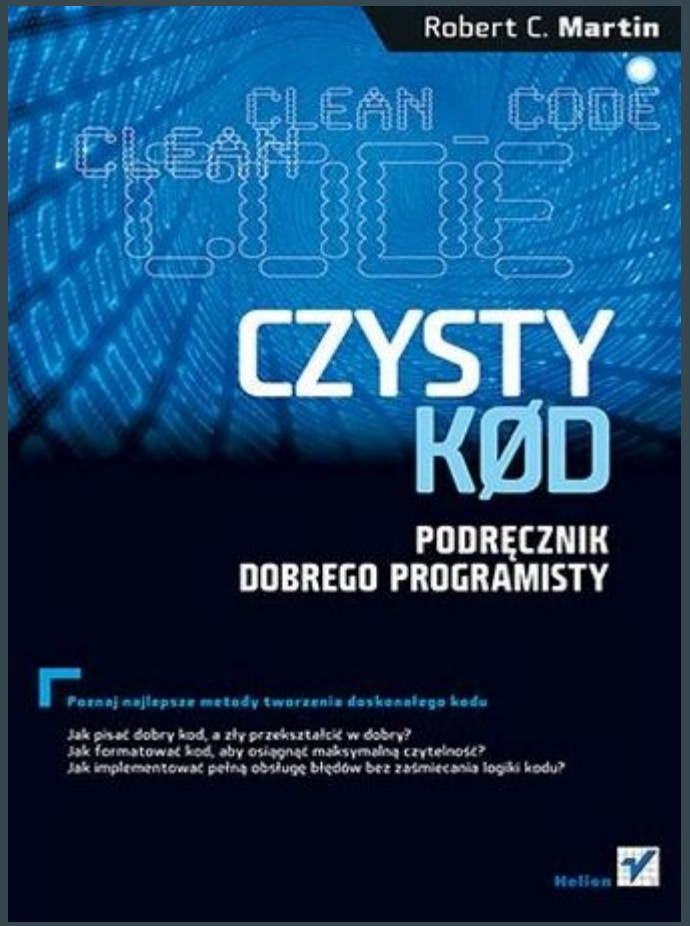
Cele

- Poszerzamy perspektywę odnośnie programowania
- Rozumiemy proces wytwarzania oprogramowania
- Testy są turbo ważne i kropka
- Mniej więcej czujemy wymagania funkcjonalne projektu

Nastawienie do problemów, niewiedzy oraz kodu

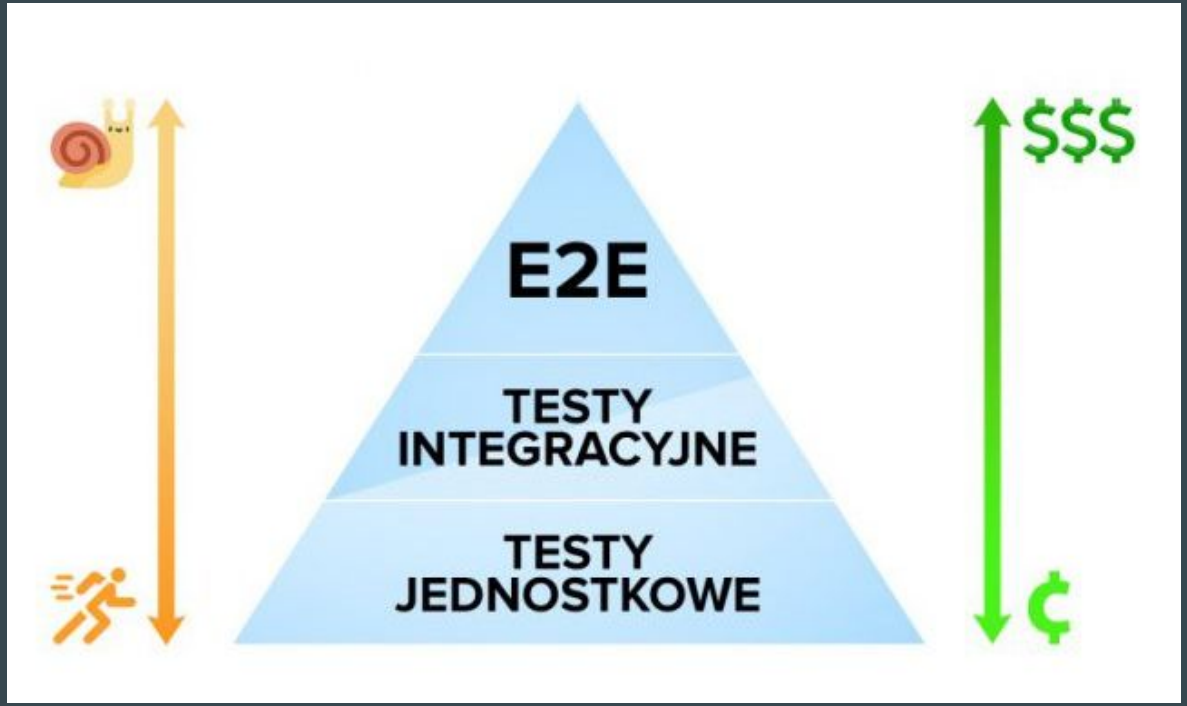
- Należy zaakceptować uczucie, że ciągle czegoś nie będziemy wiedzieć
- Można powiedzieć, że programowanie to nieustanne rozwiązywanie problemów
- Im mniej kodu tym mniej problemów :)
- Im mniej problemów tym lepiej, więc nie twórzmy ich sztucznie
- Należy wypracować metodyki poszukiwania rozwiązań (default: google, stackoverflow, książki)
- Im więcej zadanych pytań tym szybciej potwierdzamy/poprawiamy nasze hipotezy
- Tylko nieustanne "ostrzenie piły" pozwoli produkować wyższej jakości kod
- Zostało napisanych wiele książek o samej kulturze programowania


PRENTICE HALL
Helion
Software Craftsman
Profesjonalizm, czysty kod i techniczna perfekcja
Słowo wstępne Robert C. Martin
Sandro Mancuso


Robert C. Martin
CLEAN CODE
CZYSTY KØD
PODRĘCZNIK DOBREGO PROGRAMISTY
Poznaj najlepsze metody tworzenia doskonałego kodu
Jak pisać dobry kod, a zły przekształcić w dobry?
Jak formatować kod, aby osiągnąć maksymalną czytelność?
Jak implementować pełną obsługę błędów bez zaśmiecania logiki kodu?
Helion

Testy dają wysoką jakość rozwiązania

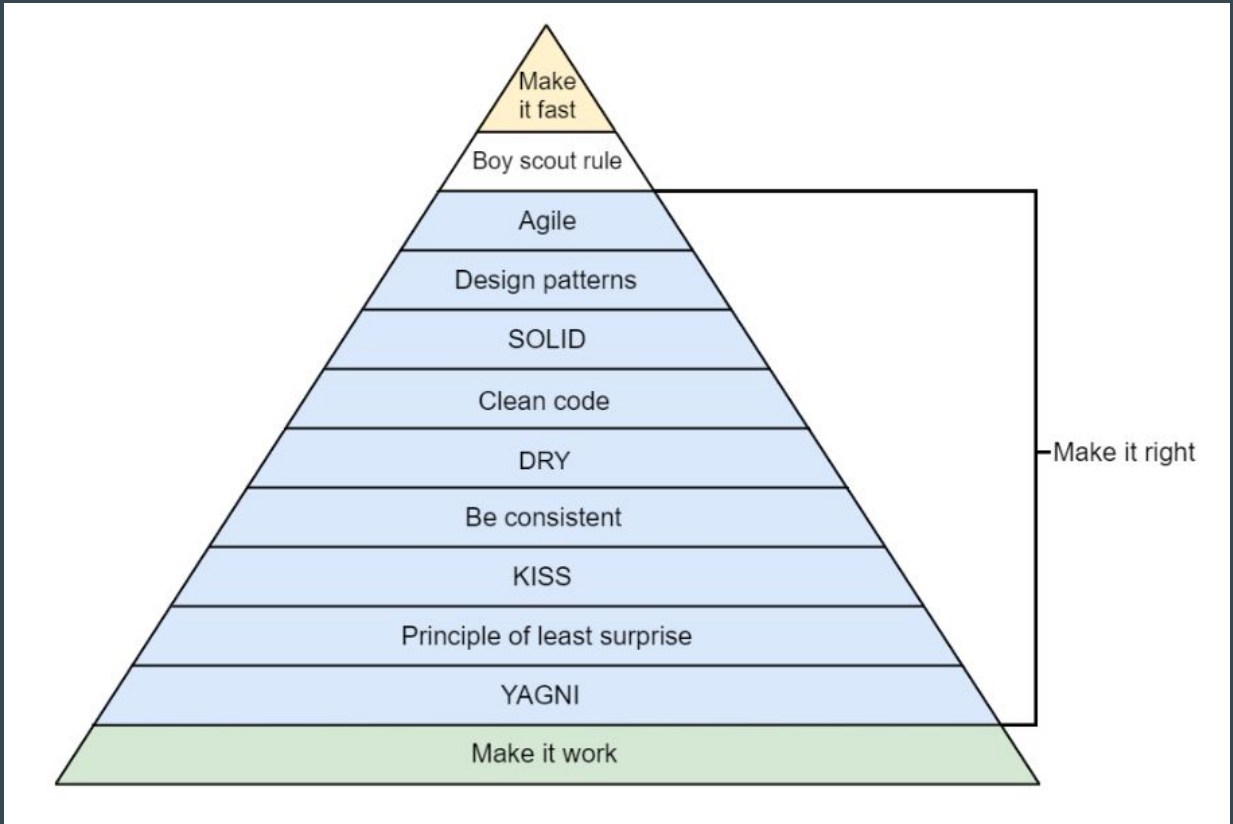
- Nieprzetestowany kod jest bezużyteczny
- Wliczajcie pisanie testów do swoich estymat jako element implementacji
- ... bo to programiści są odpowiedzialni za jakość wytworzonego kodu
- Brak testów to brak potwierdzenia, że kodzik będzie działał niezmiennie przy wprowadzaniu innych zmian
- Test coverage - procentowe pokrycie testami (ilość klas, linijek, metod)
- Wysokie pokrycie testami pozwala na zmniejszenie ryzyka przy zmianach


E2E
TESTY INTEGRACYJNE
TESTY JEDNOSTKOWE
$$$
¢

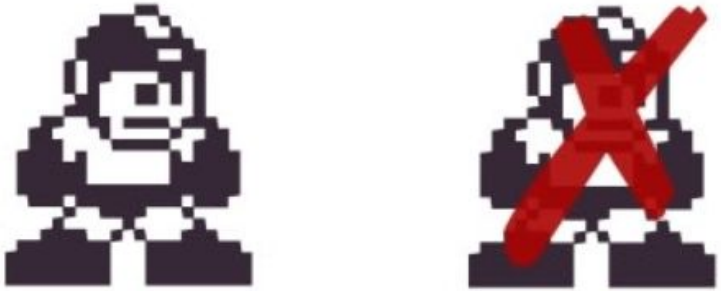
Programowanie vs kodowanie

- samodzielność realizacji zadania
- rozumienie zadania, którym się zajmujemy
- modelowanie problematyki, wizualizacja, prototypy
- inicjatywa, PoC, R&D
- współtworzenie architektury
- implementacja oparta na regułach programowania
- zarządzanie długiem technicznym
- logi oraz metryki

Reguły wytwarzania oprogramowania



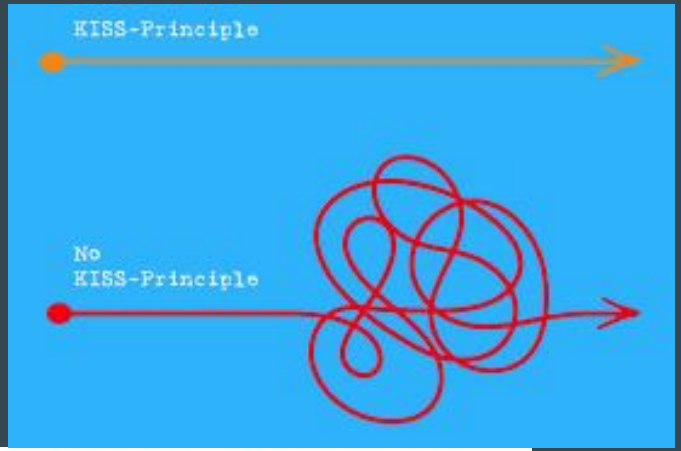
DRY: Don`t Repeat Yourself



⭐ Why DRY?

- Write code once, use it often.
- Change code in one place, see the change in all instances.
- Less code is good: It saves time and effort, is easy to maintain, and also reduces the chances of bugs.

https://dev.to/richardwynn/dry-principle-in-100-seconds-2l8c

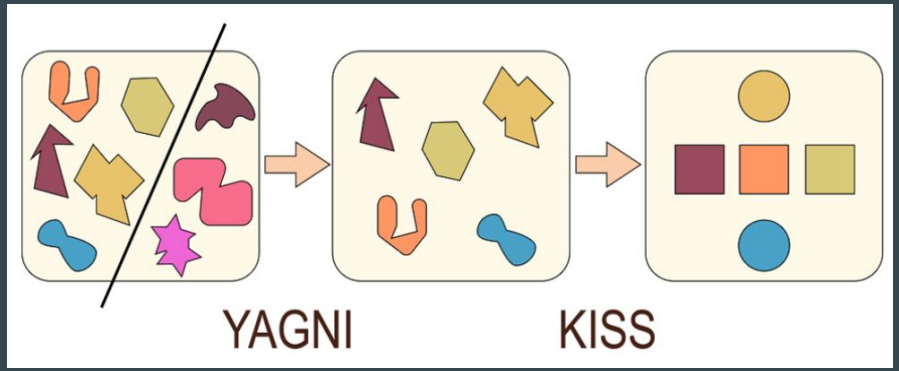


In the programming context, there are a few points to note whenever we want to reduce complexity.

- Ensure your variable names describes the variable it holds properly.
- Ensure your method names translates to the purpose of that method.
- Write comments within your method where necessary.
- Ensure your classes has a single responsibility.
- Avoid global states and behaviors like as much as you can.
- Delete instances, methods or redundant processes within the code base that are not in use.

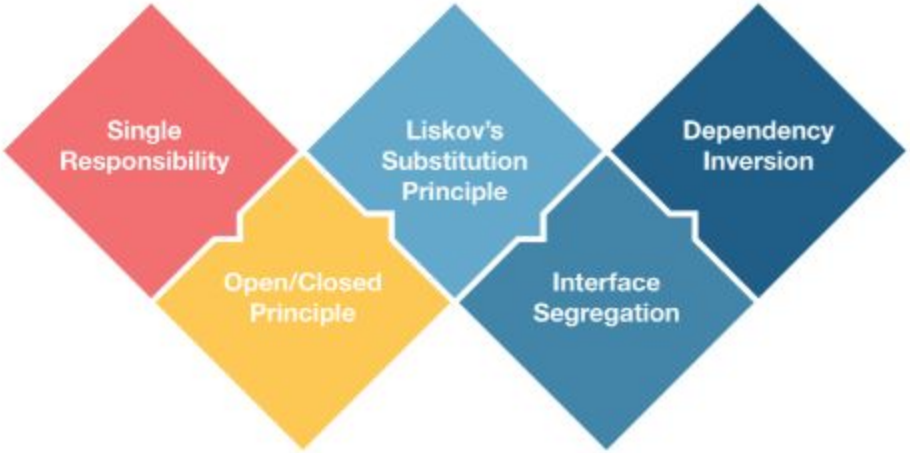
https://dev.to/kwereutosu/the-k-i-s-s-principle-in-programming-1jfg


Y.A.G.N.I.
You Ain't Gonna Need It
The best developers are the lazy ones.


YAGNI
KISS


YOU AIN'T GONNA NEED IT
Don't waste resources on what you *might* need.

S.O.L.I.D.


Single Responsibility
Open/Closed Principle
Liskov's Substitution Principle
Interface Segregation
Dependency Inversion

Projekt: Biblioteka

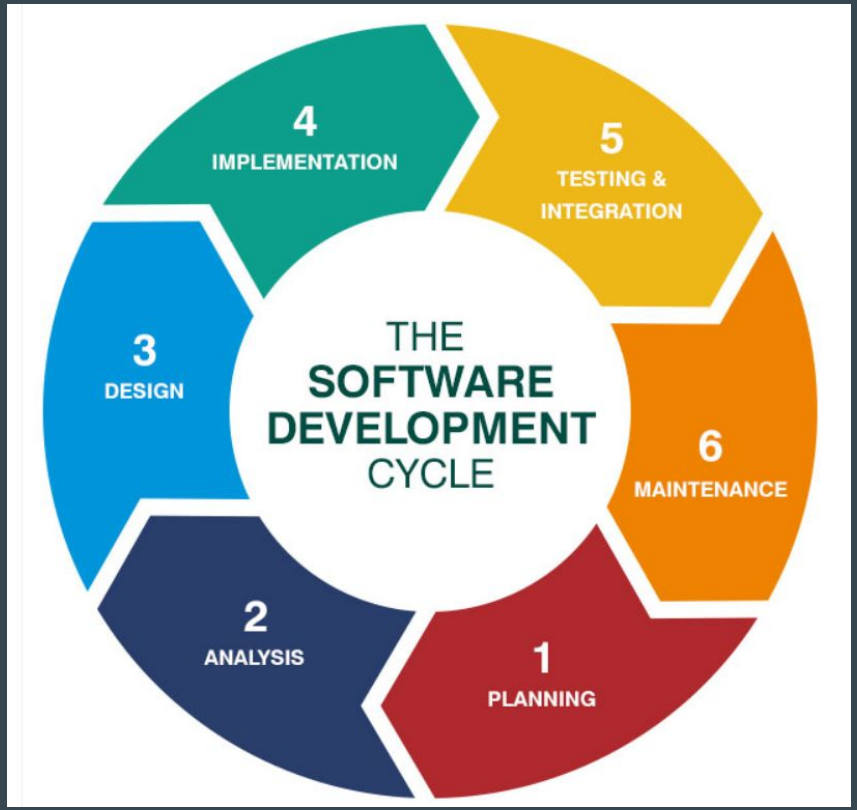

JUnit
Write a failing test
TDD
Refactor
Make the test pass
Java™
spring boot
{ REST:API }
Swagger
Supported by SMARTBEAR
Spring Data
YAGNI
KISS
DRY

Slot 2

Cele

- Cykl wytwarzania oprogramowanie
- Poznajemy sposoby wizualizacji architektury
- Poznajemy sposoby wizualizacji sekwencji zdarzeń
- Poznajemy podejście najpierw test, później kod
- Planujemy zakres prac

Cykl wytwarzania biblioteki



Analiza wymagań funkcjonalnych

TODO

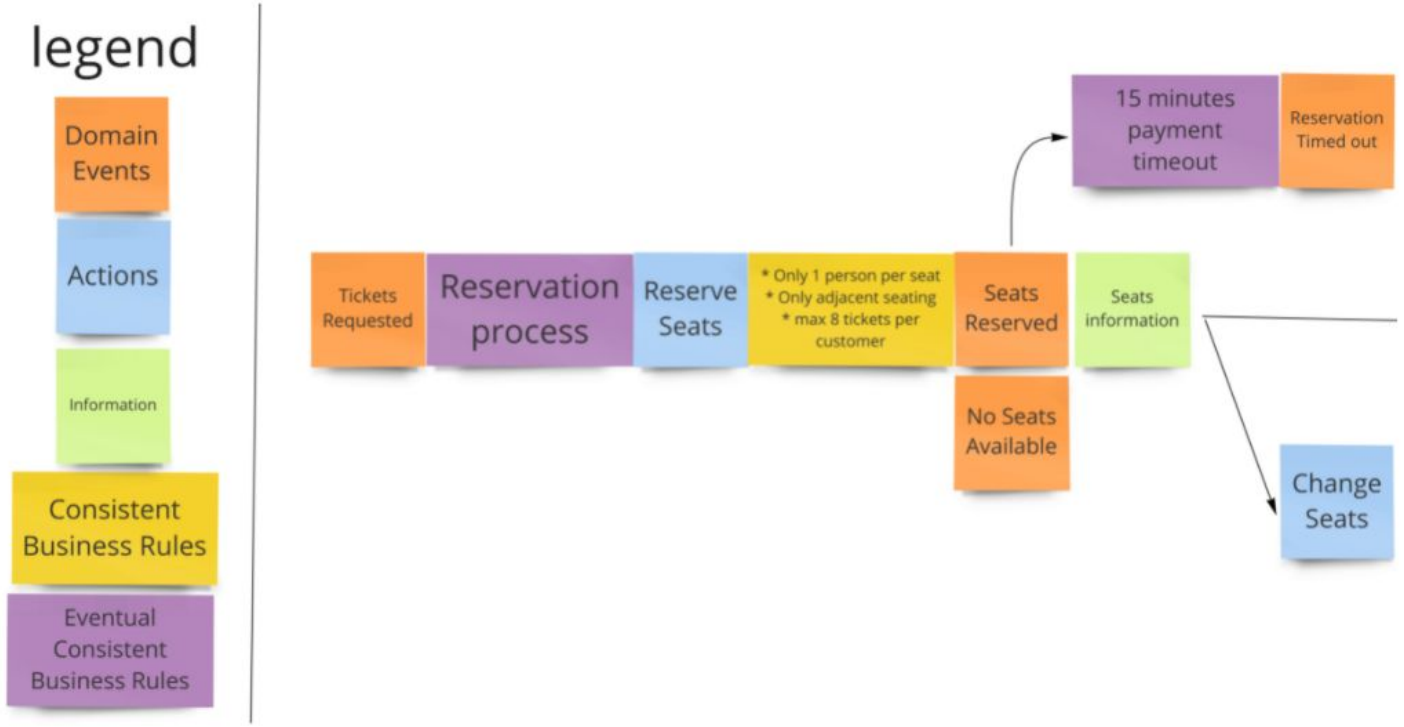
- Zapoznanie się z dokumentem
- Pogrupowanie wymagań (sesja event storming)
- Ustalenie priorytetów

TODO: Zapoznanie się z dokumentem

https://docs.google.com/document/d/1UJRAOfeiy7E4pdxeA9OjV365AsjuOYc2/edit?usp=sharing&ouid=110769739460106644580&rtpof=true&sd=true

TODO: Odkrywanie domeny + priorytety

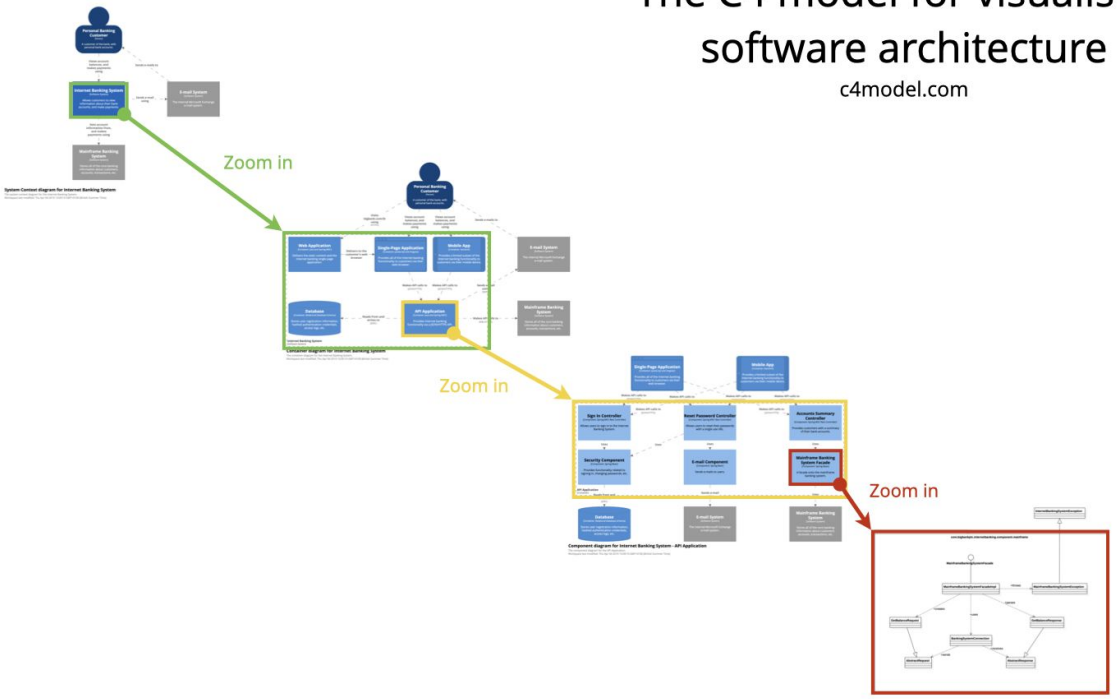
Event Storming Terms



TODO: Wizualizacja architektury

The C4 model for visualising software architecture

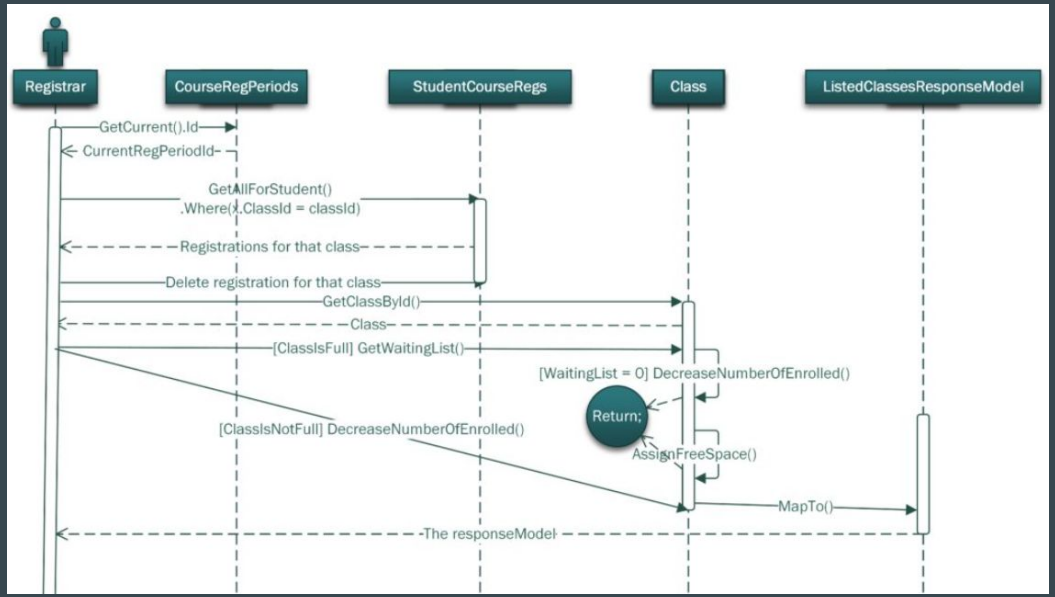
c4model.com



Level 1	Level 2	Level 3	Level 4
Context	Containers	Components	Code

TODO: Wizualizacja zależności


Unified
Modeling
Language

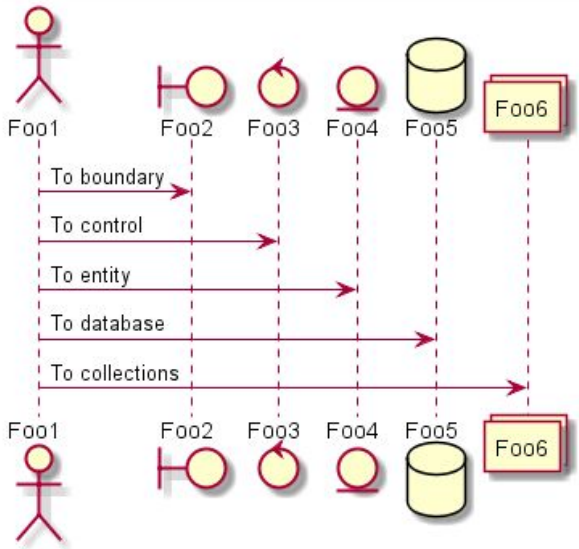

Registrar
CourseRegPeriods
StudentCourseRegs
Class
ListedClassesResponseModel
GetCurrent().Id
CurrentRegPeriodId
GetAllForStudent()
.Where(x.ClassId = classId)
Registrations for that class
Delete registration for that class
GetClassById()
Class
[ClassIsFull] GetWaitingList()
[WaitingList = 0] DecreaseNumberOfEnrolled()
Return;
[ClassIsNotFull] DecreaseNumberOfEnrolled()
AssignFreeSpace()
MapTo()
The responseModel

PlantUML

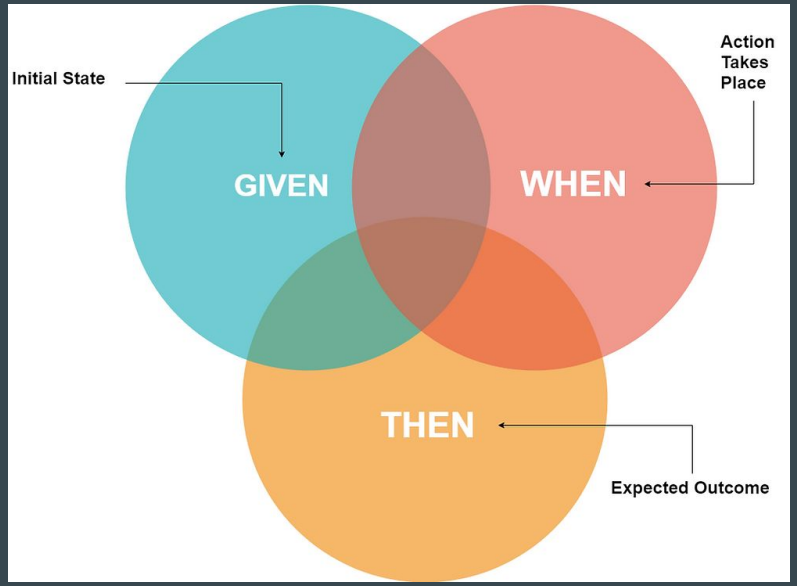
Edit online

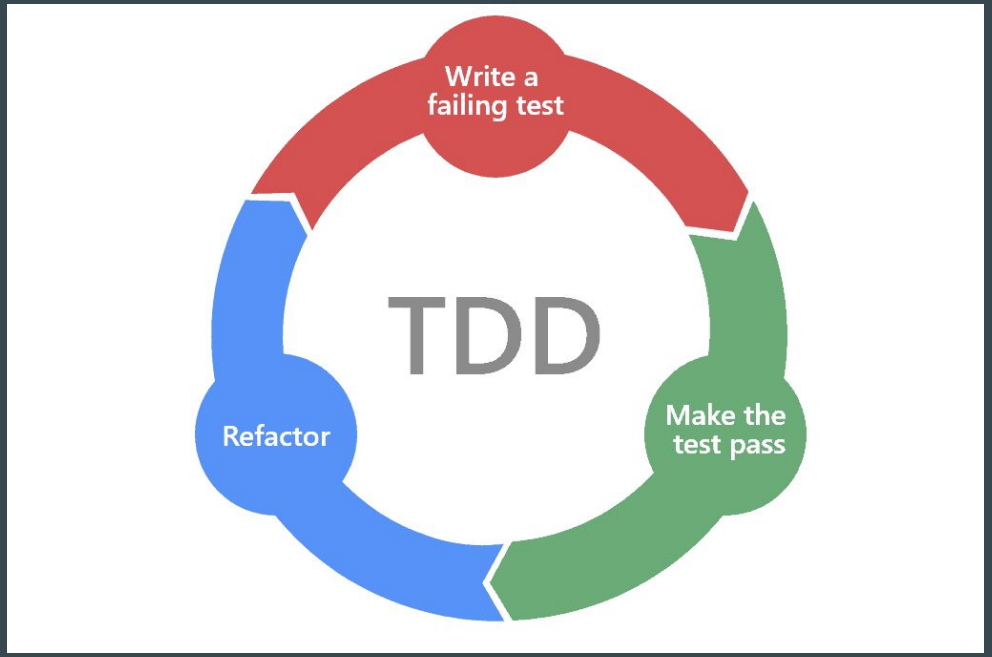
```
@startuml
actor Foo1
boundary Foo2
control Foo3
entity Foo4
database Foo5
collections Foo6
Foo1 -> Foo2 : To boundary
Foo1 -> Foo3 : To control
Foo1 -> Foo4 : To entity
Foo1 -> Foo5 : To database
Foo1 -> Foo6 : To collections

@enduml
```



Test-driven development





TODO: Zaczynamy implementację

Slot 3

Cele

- Kontynuujemy implementację ustalonego zakresu
- Zwracamy uwagę na reguły programowania
- Dbamy o jakość

Slot 4

Cele

- Aktywujemy Spring Boot
- Wprowadzamy konfigurację
- Ustawiamy odpowiedni port
- Aktywujemy akcję przy uruchomieniu aplikacji
- Aktywujemy scheduling
- Aktywujemy actuator

Slot 5

Cele

- Przypominamy sobie czym jest REST API
- Przypominamy sobie czym jest Swagger
- Aktywujemy Swaggera
- Projektujemy API
- Implementujemy API