Clean Code
Code Review

Hello!

Michał Nowakowski

Lead Software Engineer @EPAM

michal@nowakowski.me.uk

Agenda

- Wprowadzenie
- Dlaczego clean code
- SOLID DRY KISS
- Zasady clean code
- Testowanie
- Code review

Clean Code
Wprowadzenie

“Clean code always looks like it was written by someone who cares.”

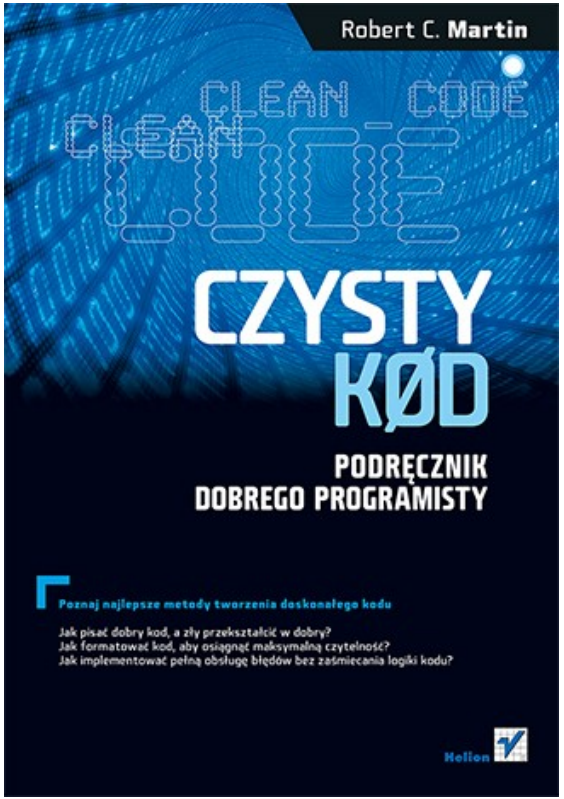
- Michael Feathers

```
7  <!-- InstanceEndEditable -->
8  <link rel="stylesheet" type="text/css" href="stylesheet.css"/>
9  <link href='http://fonts.googl                    obster' rel='styl
10 <!-- InstanceBeginEditable nam
11 <script type="text/javascript"
12 <!--
13 function MM_validateForm() { //v4.0
14   if (document.getElementById){
15     var i,p,q,nm,test,num,min,max,errors='',args=MM_validateForm.argu
16     for (i=0; i<(args.length-2); i+=3) { test=args[i+2]; val=document
17       if (val) { nm=val.name; if ((val=val.value)!="") {
18         if (test.indexOf('isEmail')!=-1) { p=val.indexOf('@');
19           if (p<1 || p==(val.length-1)) errors+='- '+nm+' must contai
20         } else if (test!='R') { num = parseFloat(val);
21           if (isNaN(val)) errors+='- '+nm+' must contain a number.\n'
22           if (test.indexOf('inRange') != -1) { p=test.indexOf(':');
23             min=test.substring(8,p); max=test.substring(p+1);
24             if (num<min || max<num) errors+='- '+nm+' must contain a
25       } } } else if (test.charAt(0) == 'R') errors += '- '+nm+' is re
26     } if (errors) alert('The following error(s) occurred:\n'+errors);
27     document.MM_returnValue = (errors == '');
28 } }
29 //-->
30 </script>
31 <!-- InstanceEndEditable -->
32 </head>
33 <body>
```

The code added just to validate one field!

```
while (menu != Menu.EXIT) {
    try {
        printOptions();
        menu = Menu.createFromInt(Integer.parseInt(input.nextLine()));
        switch (menu) {

            case SERACH_IN_BASE:
                clearScreen();
                System.out.println("Wyszukaj plik z danymi");
                SerachFundFile serachFundFile = new SerachFundFile();
                program.setPathToFile(serachFundFile.searchEngine(program.getFundsMap()));
            case FIND_GLOBAL_EXTREMES:
                clearScreen();
                //tu wstawimy poprzez metody get wartości max i min

                break;
            case FIND_LOCALE_EXTREMES:
                clearScreen();
```

Clean Code


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

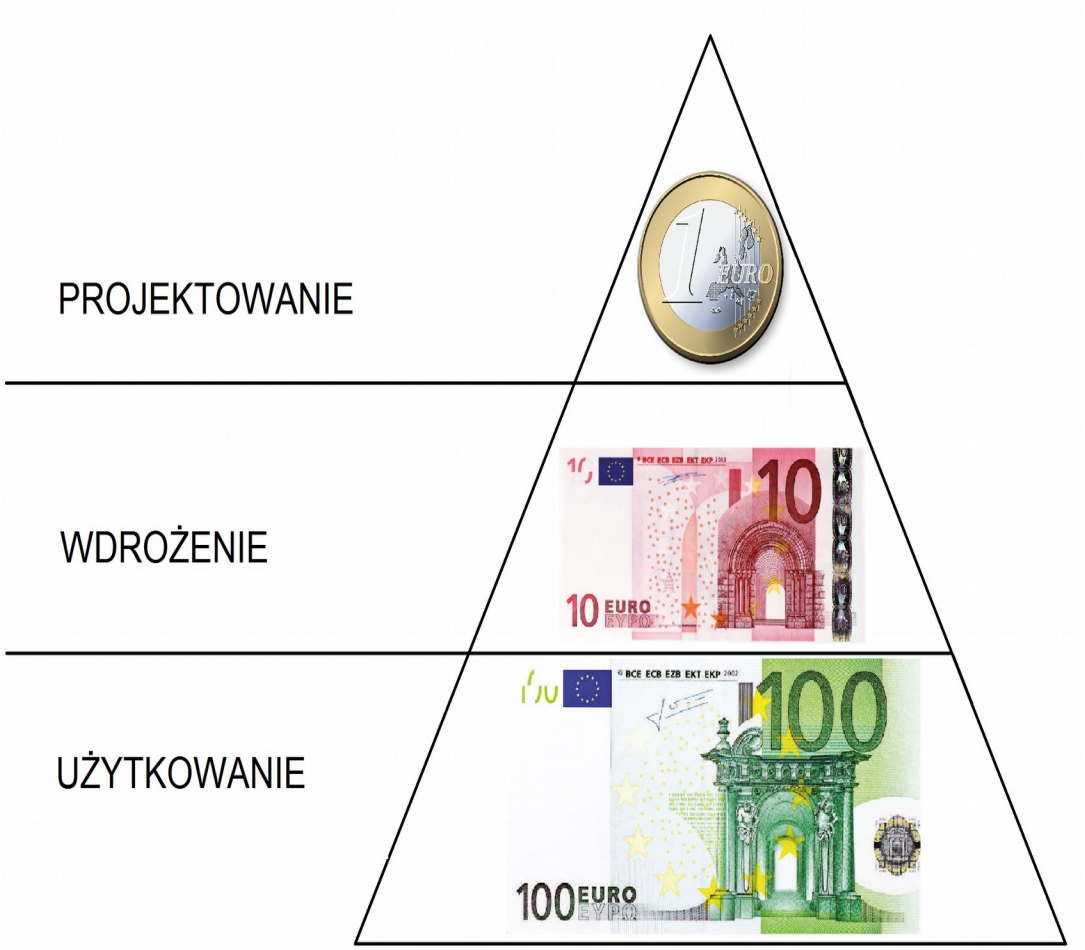

Podręcznik profesjonalnego programisty!
Mistrz czystego kodu
Kodeks postępowania profesjonalnych programistów
Helion
Robert C. Martin

Po co przejmować się jakością kodu?

Clean Code
1 – 10 – 100

- **1 – projektowanie**
  - najtańsze jest korygowanie błędów w miejscu ich potencjalnego powstania
- **10 – wdrożenie**
  - poprawianie błędów w momencie, kiedy zostaną one znalezione przez kontrolę wewnętrzną
- **100 – użytkowanie**
  - wykrycie błędu przez klienta w czasie użytkowania produktu

Clean Code
1 – 10 – 100



Clean Code
Dług technologiczny

- oznacza, że część systemu została stworzona w „prosty” sposób, bez dbania o jakość kodu
- oszczędzanie czasu i pieniędzy na rozwoju oprogramowania powoduje wzrost długu
- .. który trzeba spłacić

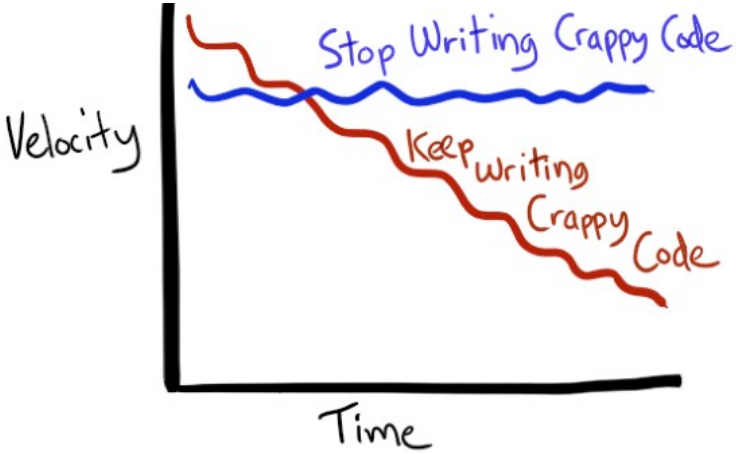
Clean Code

Dług technologiczny - powody

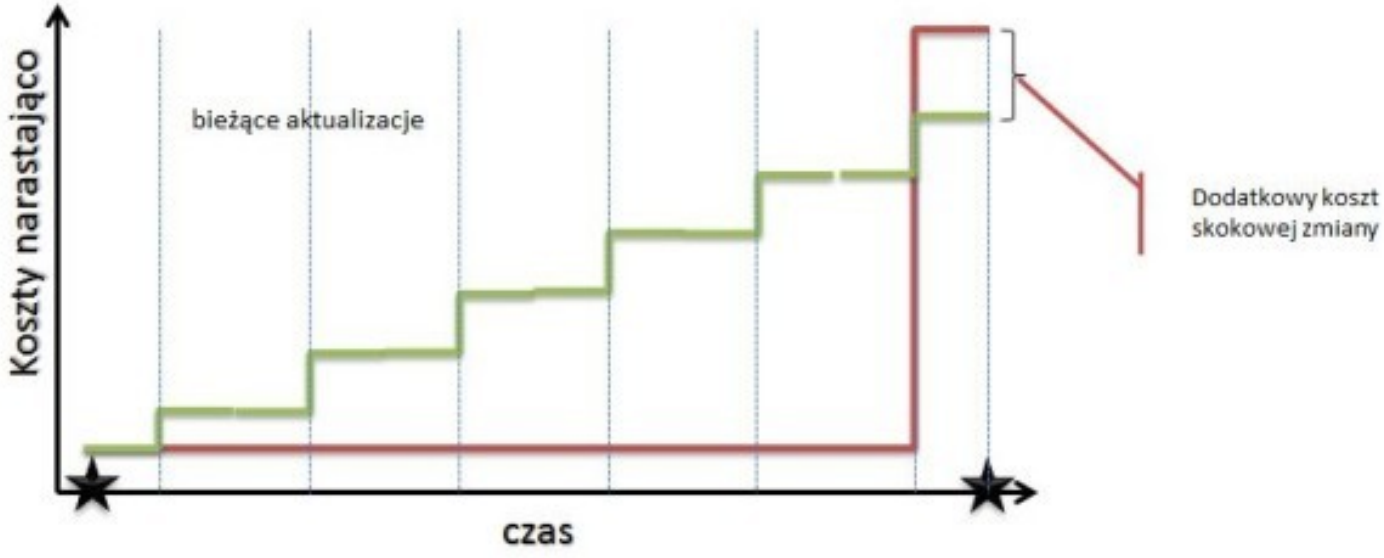
- klient chce mieć funkcjonalność ,,na wczoraj''
- … jak najtaniej
- brak wiedzy i doświadczenia
- lenistwo programistów
- brak review wykonanego zadania

Clean Code
Dlaczego warto

- mniejszy koszt w dłuższej perspektywie
- łatwiejsza modyfikacja i rozbudowa
- szybsze „wejście” nowego pracownika do projektu
- kod czytelny i zrozumiały dla każdego



Clean Code
Dlaczego warto



Clean Code
Dlaczego warto

```
public int delete(Page page) {
    if (deletePage(page) == E_OK) {
        if (registry.deleteReference(page.name) == E_OK) {
            if (configKeys.deleteKey(page.name) == E_OK) {
                logger.info("page deleted");
            } else {
                logger.info("configKey not deleted");
            }
        } else {
            logger.info("deleteReference from registry failed");
        }
    } else {
        logger.info("delete failed");
        return E_ERROR;
    }
    return E_OK;
}
```

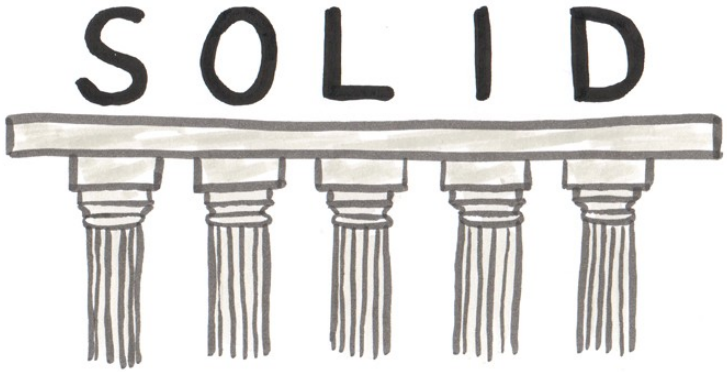
Clean Code
Dlaczego warto

```
public void delete(Page page) {
    try {
        deletePage(page);
        registry.deleteReference(page.name);
        configKeys.deleteKey(page.name);
    } catch (Exception e) {
        logger.info(e.getMessage());
    }
}
```

SOLID

Clean Code
SOLID

- **S**ingle responsibility
- **O**pen/close principle
- **L**iskov substitution
- **I**nterface segregation
- **D**ependency inversion



Clean Code
Single responsibility

- *zasada pojedynczej odpowiedzialności*
- każda klasa powinna mieć tylko jedną odpowiedzialność
- zmiana klasy powinna wynikać tylko z jednego, konkretnego powodu
- ***rób jedną rzecz i rób to dobrze***

Clean Code
Single responsibility

```
public class UserService {
    public void register(String email, String password) throws ValidationException {
        if (!email.contains("@"))
            throw new ValidationException("Email is not valid");
        User user = new User(email, password);
        database.save(user);
        smtpClient.send(new MailMessage( s1: "myEmail@myProviedr.com", email,  s2: "Content"));
    }
}
```

Clean Code
Single responsibility

```
public class UserService {
    public void validateEmail(String email) throws ValidationException {
        if (!email.contains("@"))
            throw new ValidationException("Email is not an email!");
    }

    public void register(String email, String password)  {
        User user = new User(email, password);
        database.save(user);
    }

    public void sendMessage(String address, String newEmail, String message) {
        smtpClient.send(new MailMessage(address, newEmail, message));
    }
}
```

Clean Code
Single responsibility

```
public class EmailService {
    public void validateEmail(String email) throws ValidationException {
        if (!email.contains("@"))
            throw new ValidationException("Email is not an email!");
    }

    public void sendEmail(String address, String newEmail, String message) {
        smtpClient.send(new MailMessage(address, newEmail, message));
    }
}

public class UserService {
    public void register(String email, String password)  {
        User user = new User(email, password);
        database.save(user);
    }
}
```

Clean Code
Single responsibility

```
public class EmailService {
    public boolean validateEmail(String email) {
        return email.contains("@");
    }

    public void sendEmail(MailMessage mailMessage) {
        smtpClient.send(mailMessage);
    }
}

public class UserService {
    public void register(String email, String password)  {
        User user = new User(email, password);
        database.save(user);
    }
}
```

Clean Code
Open/close principle

- zasada otwarte/zamknięte
- klasy powinny być otwarte na rozbudowę
- … ale zamknięte dla modyfikacji
- musi istnieć łatwy sposób rozbudowy modułu
- rozbudowa nie może powodować zmiany istniejącego kodu
- umiejętne wykorzystanie kompozycji, dziedziczenia i modyfikatorów dostępu

Clean Code
Open/close principle

```
public class Calculator {
    public double add(double a, double b) { return a + b; }

    public double sub(double a, double b) { return a - b; }

    public double multiply(double a, double b) { return a * b; }

    public double div(double a, double b) { return a / b; }


public class ExtendedCalculator extends Calculator {
    public double sqrt(double a) { return Math.sqrt(a); }
}
```

Clean Code
Liskov substitution

- zasada podstawienia Liskova

- korzystanie z funkcji klasy bazowej musi być możliwe również w przypadku podstawienia instancji klas pochodnych

```
public void duckyLakeExample() {
    Duck duck = new Duck();
    //ElectricDuck extends Duck
    ElectricDuck eDuck = new ElectricDuck();

    duckyLake.swim(duck);
    duckyLake.swim(eDuck);

}
```

Clean Code
Interface segregation

- zasada segregacji interfejsów
- wiele dedykowanych interfejsów jest lepsze niż jeden ogólny
- wyeliminowanie nieporęcznych, niepotrzebnie rozbudowanych interfejsów

Clean Code
Interface segregation

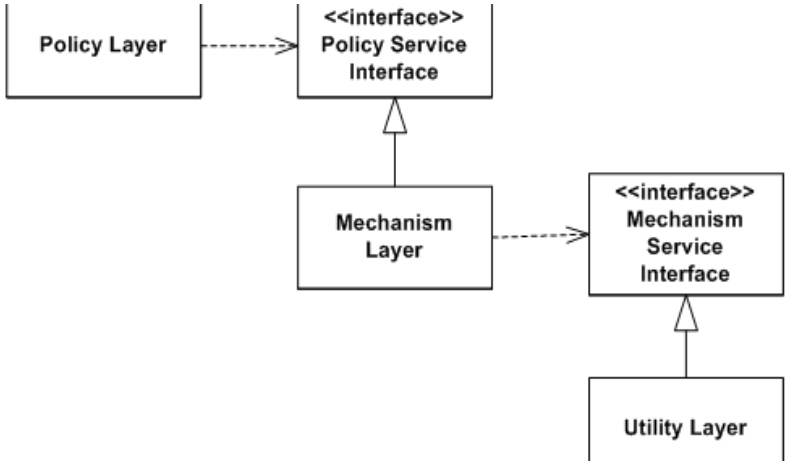
```
public interface ObjectFormatter {
    byte[] toPDF(Object someObject);

    String toXML(Object someObject);

    String toJSON(Object someObject);
}
```

Clean Code
Interface segregation

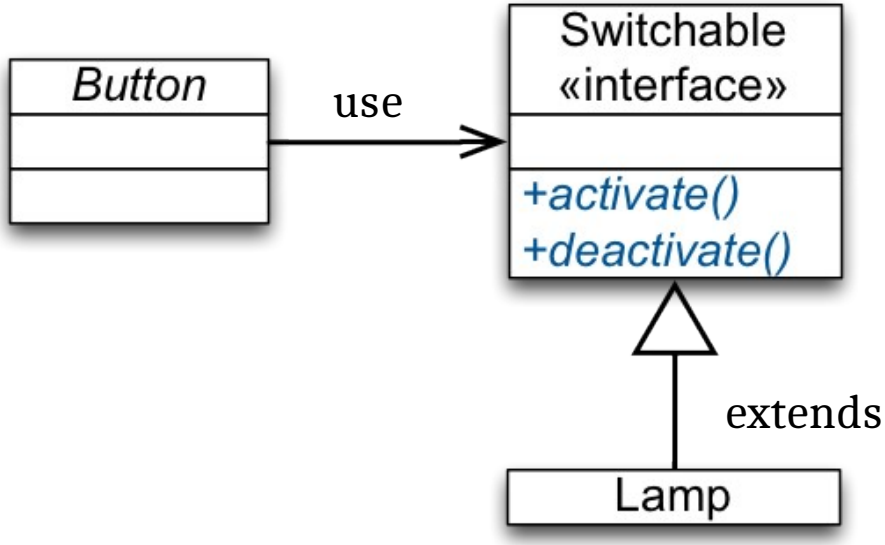
```
public interface PDFFormatter {
    byte[] toPDF(Object someObject);
}

public interface XMLFormatter {
    String toXML(Object someObject);
}

public interface JSONFormatter {
    String toJSON(Object someObject);
}
```

Clean Code
Dependency inversion

- zasada odwrócenia zależności
- wysokopoziomowe moduły nie powinny zależeć od modułów niskopoziomowych
- zależności między nimi powinny wynikać z abstrakcji



Clean Code
Dependency inversion



DRY KISS

Clean Code
DRY

- *Don’t Repeat Yourself!*
- modyfikacja w duplikowanym kodzie musiałaby nastąpić we wszystkich jego wystąpieniach

“Duplication may be the root of all evil in software”

- Robert C. Martin

Clean Code
DRY

```
public void method1() {
    //something
    sender.send( address: "administration@mail.com", value);
}

public void method2() {
    //something else
    sender.send( address: "administration@mail.com", value);
}

public void method3() {
    //something elser than else
    sender.send( address: "administration@mail.com", value);
}
```

Clean Code

DRY

```
private static String ADMIN_EMAIL = "administration@mail.com";

public void method1() {
    //something
    sendMailToAdmin();
}

public void method2() {
    //something else
    sendMailToAdmin();
}

public void method3() {
    //something elser than else
    sendMailToAdmin();
}

private void sendMailToAdmin() {
    sender.send(ADMIN_EMAIL, value);
}
```

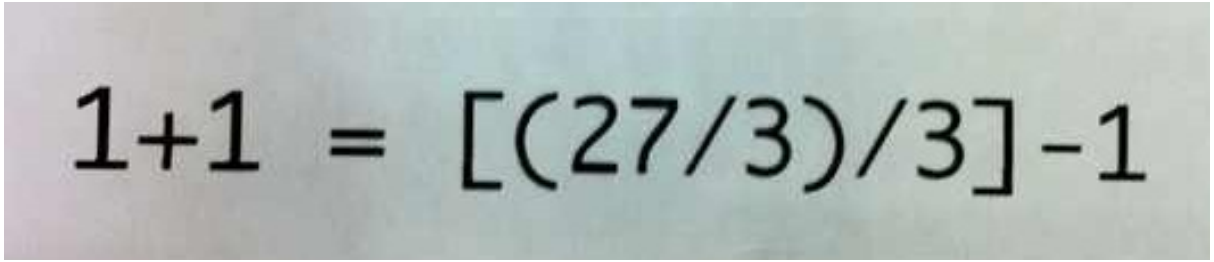
Clean Code
DRY

```
public void bar(){
  foo("A");
  foo("B");
  foo("C");
}
```

```
public void bar(){
  String [] elements = {"A", "B", "C"};

  for(String element : elements){
    foo(element);
  }

}
```

Clean Code
KISS

- *Keep It Simple, Stupid – nie komplikuj, głuptasku*
- zwiększenie jakości i czytelności kodu
- łatwiejsze zrozumienie i utrzymanie



Clean Code
KISS

```
public Boolean isSuperUser() {
    Boolean result;

    if (isAuthenticationCorrect()) {
        result = true;
    }
    else {
        retult = false;
    }

    return result;
}
```

```
public Boolean isSUperUser() {
    return isAuthenticationCorrect();
}
```

Clean Code
YAGNI

- *You Ain't Gonna Need It*
- nie należy pisać kodu „na zapas", który nie jest potrzebny
- „może się kiedyś przydać"
- ładny kod, który nic nie robi

Clean Code
YAGNI

```
public void saveUser(User user) {
    entityManager.save(user);
}

/*
//will probably need this later
 public void updateUser(User user) {
     entityManager.update(user);
 } */
```

Clean Code
Przewidywalność

- *Principle of least astonishment*
- wynik danej operacji/metody powinien być:
  - oczywisty
  - przewidywalny
  - zgodny z nazwą operacji

Clean Code
Przewidywalność

```
public int multiply(int a, int b) {
    return a * b;
}
```

Clean Code
Przewidywalność

```
public class UserService {
    public void register(String email, String password) throws ValidationException {
        if (!email.contains("@"))
            throw new ValidationException("Email is not valid");
        User user = new User(email, password);
        database.save(user);
        smtpClient.send(new MailMessage( s1: "myEmail@myProviedr.com", email,  s2: "Content"));
    }
}
```

Clean Code

Zasada skauta

Clean Code
Zasada skauta

- *zawsze zostawiaj obozowisko czystsze niż je zastałeś*
- zastany w kodzie bałagan warto posprzątać
- nawet jeśli to nie nasz nasz kod

Clean code w praktyce

Clean code
Nazwy

- nazwy klas/metod/pól powinny jednoznacznie określać do czego służą
- powinny być znaczące i opisujące intencje
- mogą pochodzić z terminologii informatycznej
  np. *bubbleSort()*

Clean code
Nazwy

```
private String qwe;
private User u;

public void method() {
    //something
}
```

Clean code
Komentarze

- unikamy zbędnych komentarzy
- właściwie to unikamy wszystkich komentarzy
- zwykle nazwa powinna wystarczyć do zrozumienia
- potrzeba napisania komentarza to potrzeba przeorganizowania kodu

“Every time you write a comment, you should grimace and feel the failure of your ability of expression.” - Robert C. Martin

Clean code
Komentarze

- zamiast komentować, użyj do tego funkcji lub zmiennej

```
//check if the employee should get extra bonus
if (employee.getType().equals(HOURLY) && employee.getAge() > 55) {
//...
}

if (isEligibleForBonus(employee)) {
//...
}
```

Clean code
Zasady

- metody nie powinny kłamać
- powinny wykonywać to i tylko to na co wskazuje ich nazwa
- liczba argumentów metody nie powinna przekraczać 3
- gdy potrzeba przekazać więcej parametrów, to przekazujemy obiekt

```
public void printCircle(int x, int y, int r, Color color) {...}
```

Clean code
Zasady

```
private class Circle {
    private int x;
    private int y;
    private int r;
    private Color color;
}
```

```
public void printCircle(Circle circle) {...}
```

Clean code
Zasady

- nie „hardcodujemy” zmiennych – wyciągamy wszystko do pól *static final*
- unikamy przekazywania *null*
- metoda nie powinna przyjmować ani zwracać *null*
- wykorzystujemy Optional albo puste kolekcje

```
public List<String> getNames() {
    if (names == null) {
        return new ArrayList<>();
    }

    return names;
}
```

Clean code
Zasady

- każdy pull request powinien zostać poddany pod review drugiego programisty
- nawet wtedy, gdy kod pisze *senior*, a review robi *junior*
- nauka i poznawanie nowych funkcji przez osobę robiącą review

Clean code
Zasady

- fragmenty, które są tworzone jak „buildery” (np. toString) tworzymy wielolinijkowo

```
public String toString() {
    return "Name: '"
            + this.name
            + "', Height: '"
            + this.height
            + "', Birthday: '"
            + this.bDay + "'";
}
```

Clean code
Zasady

- brzydki wizualnie kod = zła koncepcja
  np. for z 4 poziomami zagłębienia
- prawie zawsze da się coś napisać lepiej
- najpierw myślenie i analiza, potem kodowanie
- *dobry programista zaczyna pracę od kartki i ołówka*

Clean code
Zasady

- jeżeli metoda może zakończyć się warunkowo przed jej zakończeniem, to warunek ten powinien być sprawdzany jak najwcześniej

```
if (isInvalid() {
    return;
} else {
    //...
}
```

Clean code
Zasady

- *step-down rule* – kod czytamy od góry do dołu

```
private static void prepareDocument() {
    prepareHeader();
    prepareBody();
    prepareFooter();
}

private static void prepareHeader() {...}

private static void prepareBody() {...}

private static void prepareFooter() {...}
```

Clean code
Zasady

- nie bój się używać długich nazw jeśli to konieczne
- chociaż im krótsze tym lepsze
- warto poświęcić czas na wymyślenie dobrej nazwy
- trzymaj się konwencji używanej w projekcie

Clean code
Zasady

```
public void saveButtonController() {};

public void dealCancelButton() {};
```

Clean code
Zasady

- wcięcia, wcięcia, wcięcia!
- klamra { na końcu wiersza, nie od nowej linii
- odstępy między parametrami

```
method(paramA, paramB)  {
        for (int i = 0; i < 5; i++) {
                if (value == null) {
                        //something
                }
        }
}
```

Clean code
Obsługa błędów

```
DeviceHandle handle = getHandle(DEV1);
// Check the state of the device
if (handle != DeviceHandle.INVALID) {
        // Save the device status to the record field
        retrieveDeviceRecord(handle);
        // If not suspended, shut down
        if (record.getStatus() != DEVICE_SUSPENDED) {
                pauseDevice(handle);
                clearDeviceWorkQueue(handle);
                closeDevice(handle);
        } else {
                logger.log("Device suspended. Unable to shut down")
        }
```

Clean code
Obsługa błędów

```
public void sendShutDown() {
     try {
               tryToShutDown();
     } catch (DeviceShutDownError e) {
               logger.log(e);
     }
 }

 private void tryToShutDown() throws DeviceShutDownError {
         DeviceHandle handle = getHandle(DEV1);
         DeviceRecord record = retrieveDeviceRecord(handle);
```

Clean code
Mental mapping

```
for (e = 0; e < 100; e++) {
        for (p = 0; p < 100; p++) {
                // ...
        }
}
```

```
for (i = 0; i < 100; i++) {
        for (j = 0; j < 100; j++) {
                // ...
        }
}
```

Clean code
Zasady

- ciała metod powinny być małe
- a nawet jeszcze mniejsze
- robić dokładnie to o czym mówi ich nazwa
- i robić to dobrze

```
public Double muliply(Double value, Double multiplicator) {
    return value * multiplicator;
}
```

```java
public void savingMethod(User user) {
    String name = "name";

    if (!user.validateName()) {
        return;
    }

    switch (user.getGroupId()) {
        case 1 :
            //doSomeCode
            break;

        case 2 :
            //another code
            break;
    }

    user.setSavingName(name);

    entityManager.save(user);
}
```

Testowanie

Clean code
Testowanie

- redukcja błędów
- redukcja kosztów modyfikacji
- obrona przed innymi programistami
- wbrew pozorom przyspiesza proces kodowania

Clean code
Testowanie

- ✔ szuflada1
- ✔ szuflada2
- ✔ szafka1
- ✔ szafka2

- ✔ testy zakończone sukcesem

Clean code
Zasady

- pisząc testy jednostkowe dzielimy je na sekcje
  - *//GIVEN* – co mamy dane
  - *//WHEN* – co się dzieje
  - *//THEN* – jaki jest oczekiwany efekt
- **nie** wszystkie sekcje muszą się zawierać w teście

Clean code
Zasady

- nazwa testu może być długa, musi jednoznacznie określać co test sprawdza

```
@Test
public void multiplyValuesWhenValueIsNull() {...}

@Test
public void multiplyValuesWhenValuesIsIncorrectType() {...}
```

Clean code
Zasady

```
@Test
public void multiplyWhenValuesAreCorrect() {
    //GIVEN
    Double value = 3.6;
    Double multiplicator = 2.0;
    Double result;
    Double expectedResult = 7.2;

    //WHEN
    result = multiply(value, multiplicator);

    //THEN
    assertEquals(expectedResult, result);
}
```

Clean code
Testowanie - korzyści

- pewność, że kod działa poprawnie
- szybka weryfikacja poprawności wprowadzanych zmian
- dodatkowa dokumentacja projektu
- pewność, że nie zepsuliśmy niczego w innej części projektu

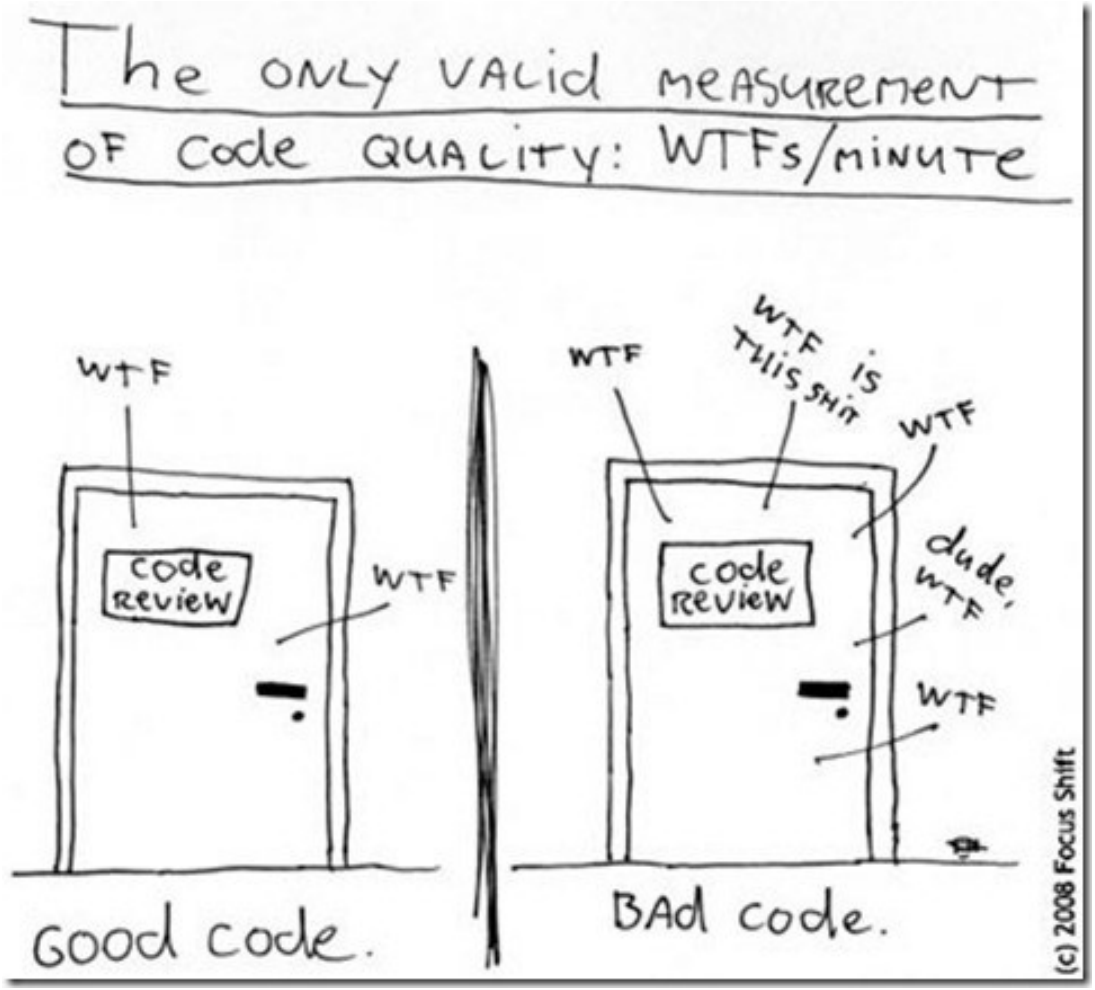
Code review

Clean code
Code review

- ma na celu wykrycie i poprawienie błędów w kodzie
- lub zwiększenie jakości poprawnie działającego kodu
- pozwala innym członkom zespołu zapoznać się ze zmianami w projekcie
- znacząco zwiększa jakość
- pułapka “formalnego review”

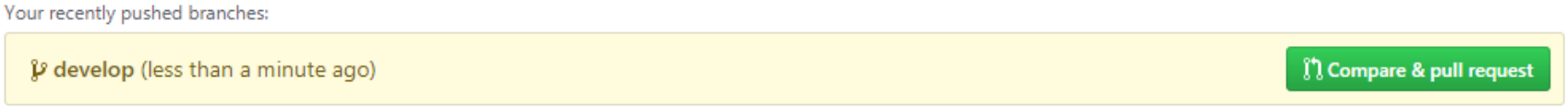
Clean code
Code review

- wykonywane przez inną osobę/osoby
- niezbędny przed mergem zmian do głównego brancha
- wykonywany przy każdym pushowaniu zmian


The ONLY VALID MEASUREMENT OF Code QUALITY: WTFs/MINUTE
WTF
code review
WTF
Good code.
WTF
WTF is THIS SHIT
WTF
code review
dude, WTF
WTF
BAd code.
(c) 2008 Focus Shift

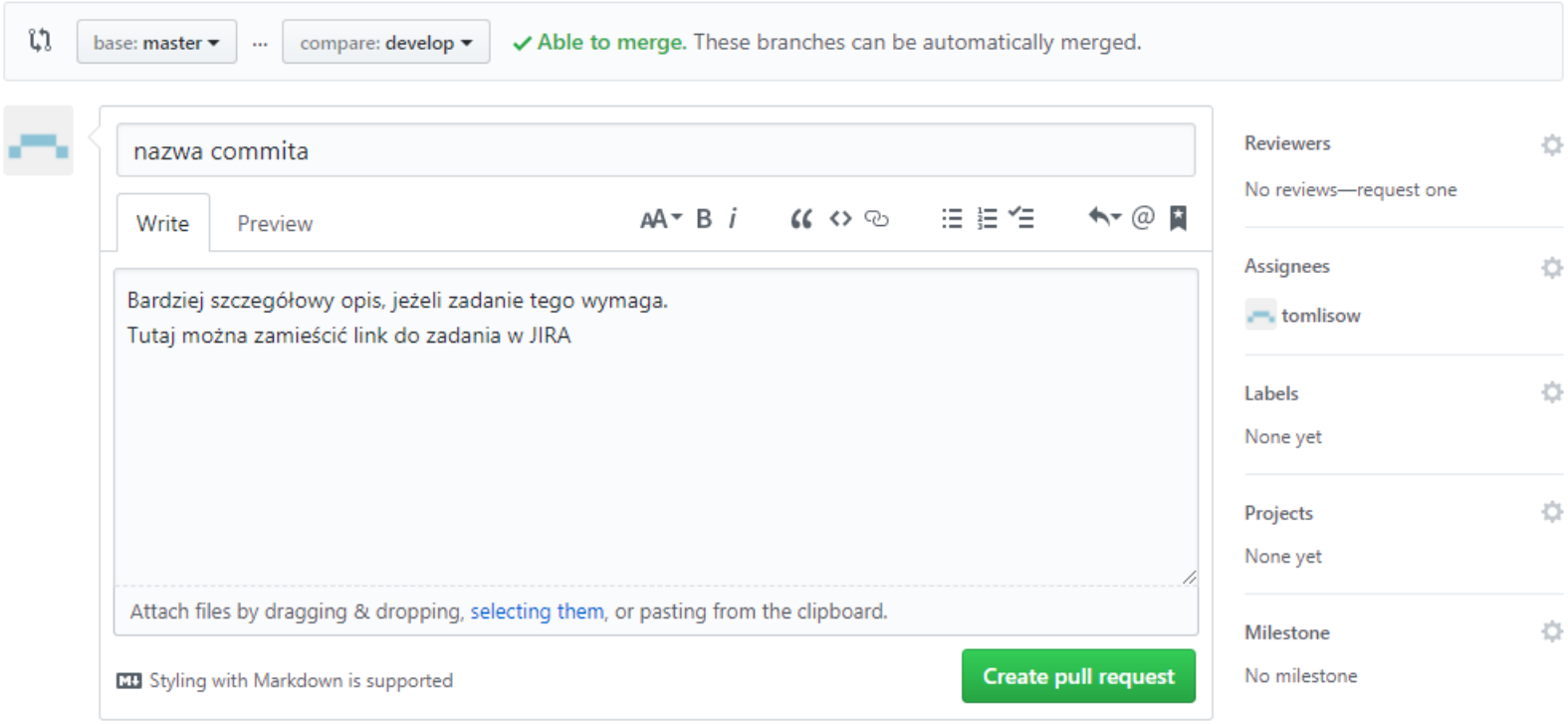
Clean code
Code review

- opcja dostępna po wykonaniu pusha
- wchodzimy na github i klikamy



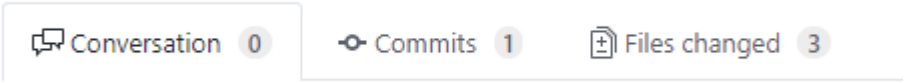
- wybieramy opcje, opisujemy co zawiera pull request oraz wołamy osobę do wykonania naszego review
- wystawiamy pull request
- można wprowadzać modyfikacje do istniejącego otwartego zgłoszenia

Clean code
Code review



Clean code
Code review

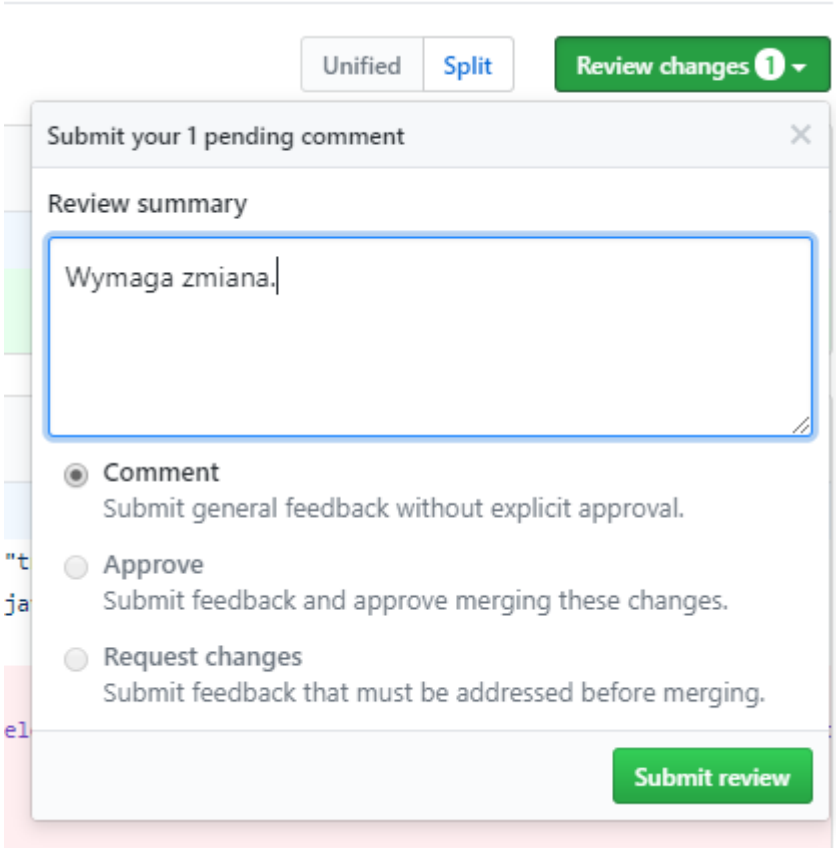
- w otwartym pull requeście możemy podejrzeć commity, zmiany w kodzie oraz prowadzić dyskusję z zespołem



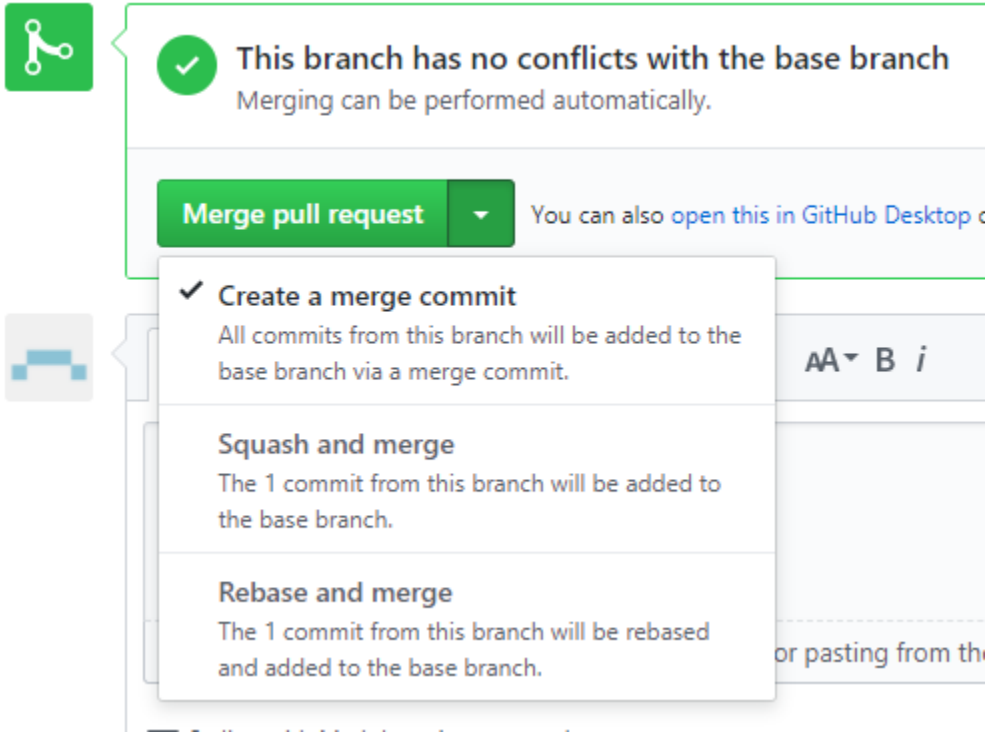
- zmiany wyrzucone z kodu są na czerwono, dodane na zielono

```
41  -    <folding>
42  -      <element signature="e#114#115#0" expanded="true" />
43  -      <element signature="e#166#167#0" expanded="true" />
44  -    </folding>
    39 + <state relative-caret-position="187">
    40 +   <caret line="11" column="40" selection-start-line="11"
    41 +   <folding />
```

Clean code
Code review



Clean code
Code review

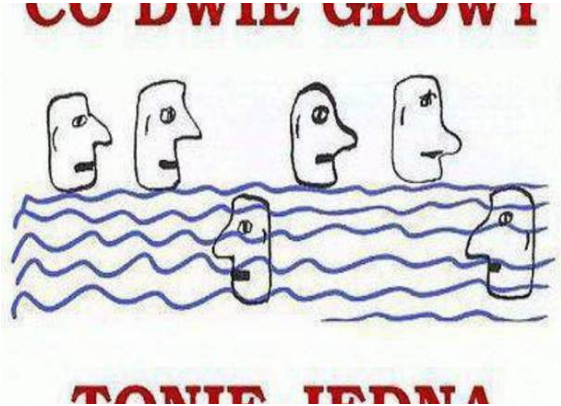


Clean code
Code review

- wybierz dowolną klasę z grupowego projektu, która wymaga refaktoringu
- wprowadź zmiany zgodnie z zasadami clean code
- wystaw Pull Request wprowadzonych zmian
- przeprowadź review zmian innych osób z zespołu

Clean code
Pair programming

- programowanie w parach
- jedna stacja robocza
- jedna osoba pisze, druga mówi co pierwsza ma pisać
- live code review



Must read

- *"Clean Code: A Handbook of Agile Software Craftsmanship"*
  - Robert Martin
- *"The Clean Coder: A Code of Conduct for Professional Programmers"*
  - Robert Martin
- *"Refactoring: Improving the Design of Existing Code"*
  - Martin Fowler
- *"Test Driven Development: By Example"*
  – Kent Beck

Thanks!!

Q&A