

Clean Code Code Review



Hello!

Tomasz Lisowski

Software developer, JIT Solutions
IT trainer

Agenda

- wprowadzenie
- dlaczego clean code
- SOLID DRY KISS
- zasady clean code
- czysty kod w praktyce
- testowanie
- code review



Clean Code

Wprowadzenie

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

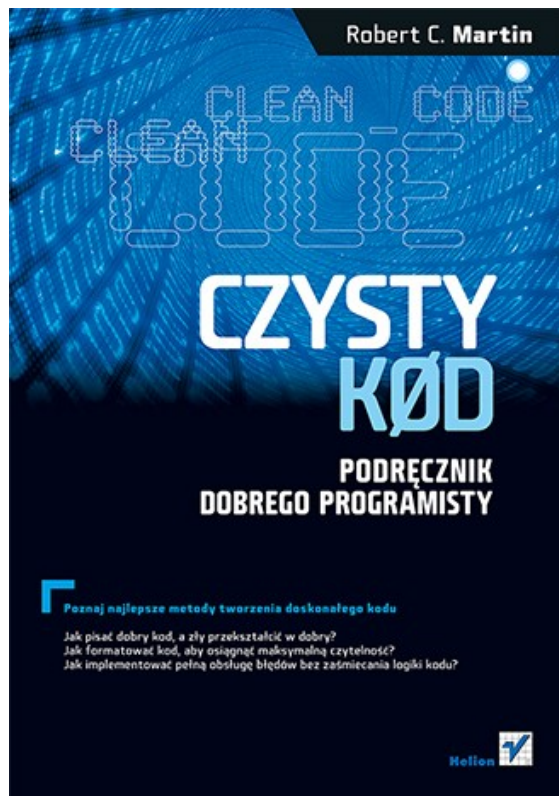
- Michael Feathers

```
7 <!-- InstanceEndEditable -->
8 <link rel="stylesheet" type="text/css" href="stylesheet.css"/>
9 <link href='http://fonts.googleapis.com/css?family=Roboto:400,300,700' rel='stylesheet' type='text/css'>
10 <!-- InstanceBeginEditable name="Form" -->
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.arguments;
16         for (i=0; i<(args.length-2); i+=3) { test=args[i+2]; val=document.getElementById(test);
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 contain an email address.\n';
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 number in the range '+min+' to '+max.\n';
25                     } } } else if (test.charAt(0) == 'R') errors += '- '+nm+' is required.\n';
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



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

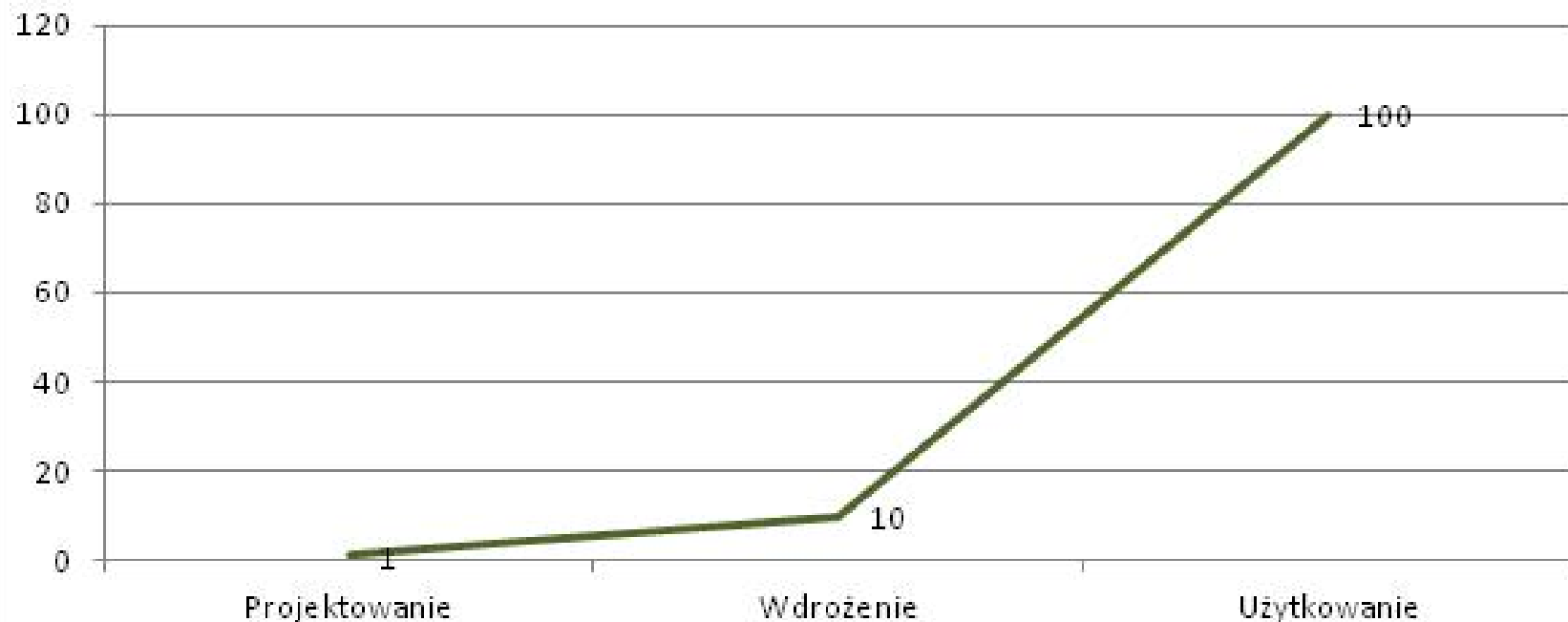
■ 100 – użytkowanie

- wykrycie błędu przez klienta w czasie użytkowania produktu

Clean Code

1 - 10 - 100

Koszty wykrycia i naprawy błędów

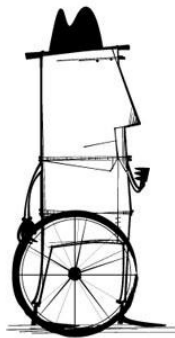


Clean Code

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

ERRR...



*CAN'T STOP.
TOO BUSY!!*



Clean Code

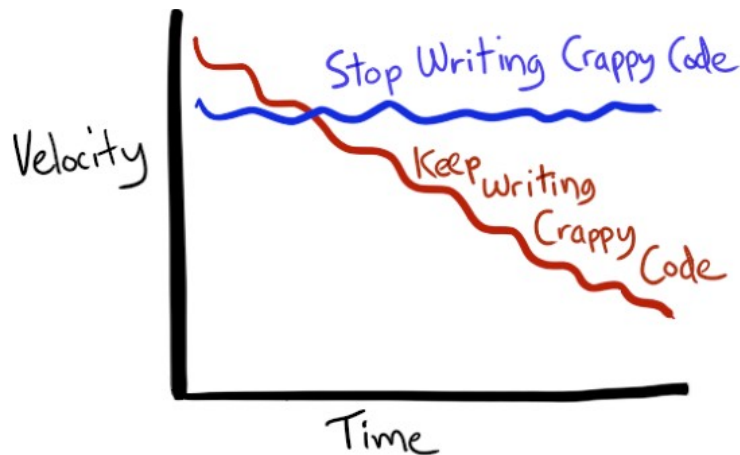
Dług technologiczny - powody

- 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

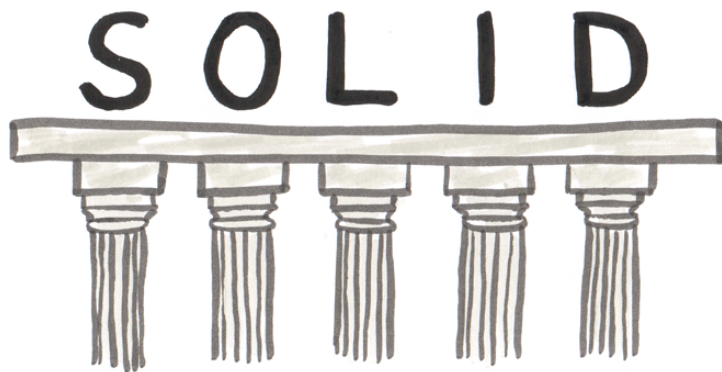


SOLID

Clean Code

SOLID

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



Clean Code

Single responsibility

- *zasada jednej odpowiedzialności*
- każda klasa powinna mieć tylko jedną odpowiedzialność
- żadna klasa nie może być modyfikowana z więcej niż jednego powodu – albo jest źle zaprojektowana



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

Clean Code

Liskov substitution

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

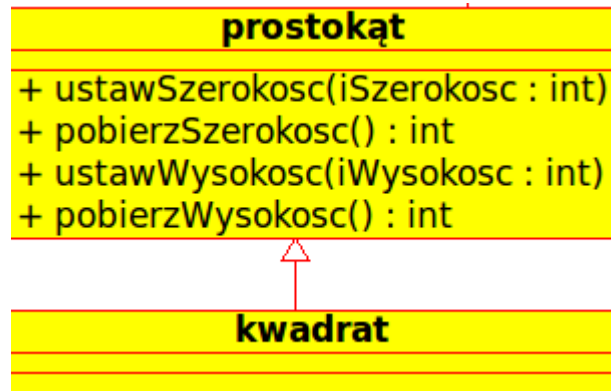
np. `pojazd.jedzProsto()`

`Samochod extends Pojazd` → `samochod.jedzProsto()`

Clean Code

Liskov substitution

- ustawSzerokosc() i ustawWysokosc() to naturalne metody dla klasy Prostokąt
- wywołanie ustawSzerokosc() dla kwadratu zmieni wartość wysokości
- łamanie zasady jednej odpowiedzialności
- korzystanie z funkcji klasy bazowej nie jest tutaj możliwe



Clean Code

Interface segregation

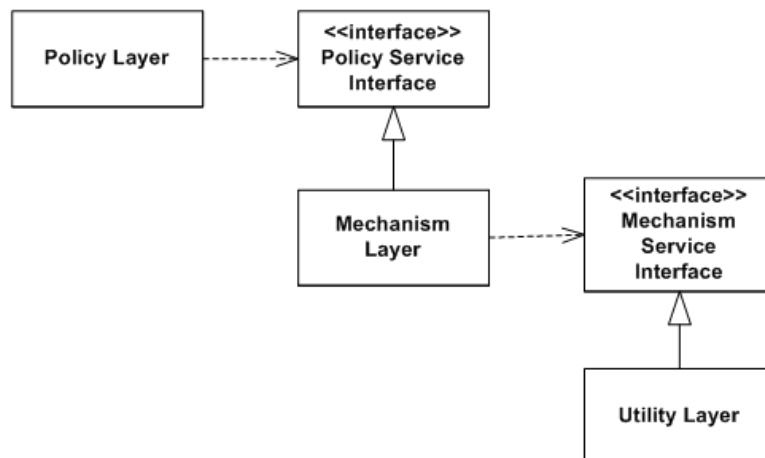
- zasada segregacji interfejsów
- wiele dedykowanych interfejsów jest lepsze niż jeden ogólny
- wyeliminowanie nieporęcznych, niepotrzebnie rozbudowanych interfejsów



Clean Code

Dependency inversion

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



DRY KISS

Clean Code

DRY

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

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

- Robert C. Martin

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

$$1+1 = [(27/3)/3]-1$$

Clean Code

KISS

```
public Boolean isSuperUser() {  
    Boolean result;  
  
    if (isAuthenticationCorrect()) {  
        result = true;  
    }  
    else {  
        result = 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

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

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

Clean code

Nazwy



Clean code

Komentarze

- unikamy zbędnych 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 (employee.isEligibleForBonus()) {  
    // ...  
}
```

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 wcześniej, to warunek ten powinien być sprawdzany jak najwcześniej

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

Clean code

Zasady

- *step-down rule* – kod czytany 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

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 multiply(Double value, Double multiplier) {  
    |   return value * multiplier;  
}
```

Testowanie

Clean code

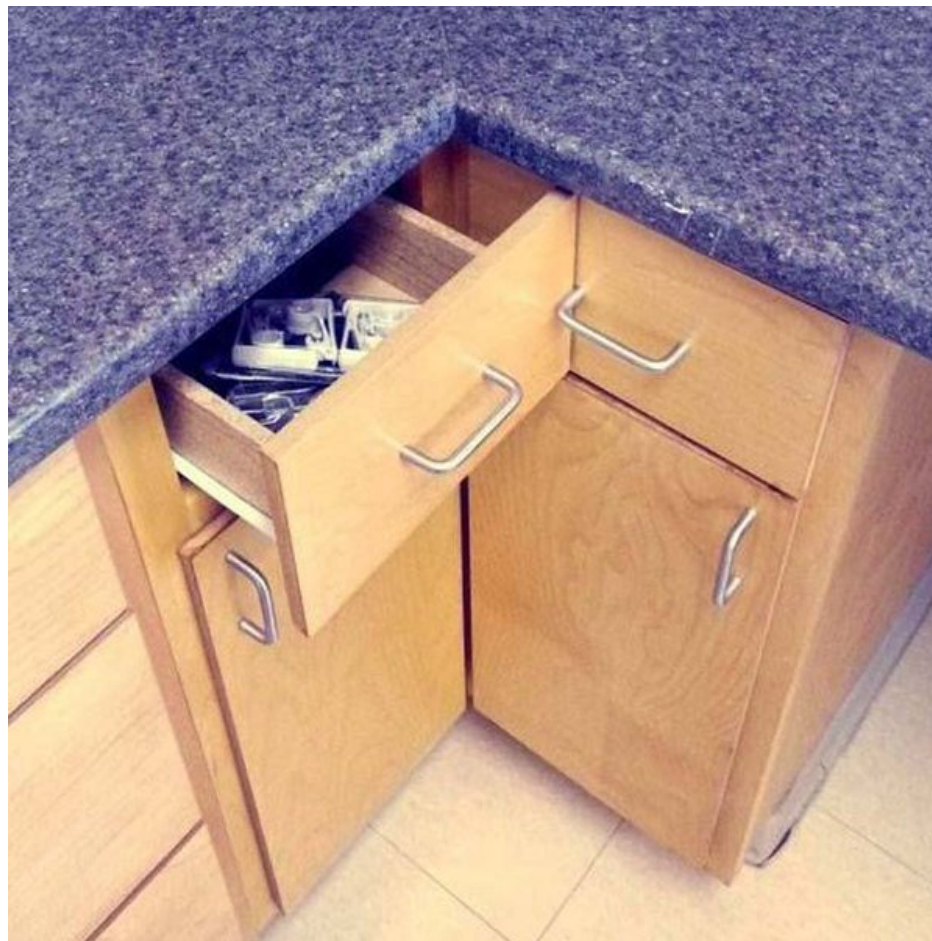
Testowanie

- 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
- 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 = 2.0;
    Double valueNull = 3.6;
    Double result;
    Double expectedResult = 7.2;

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

    //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

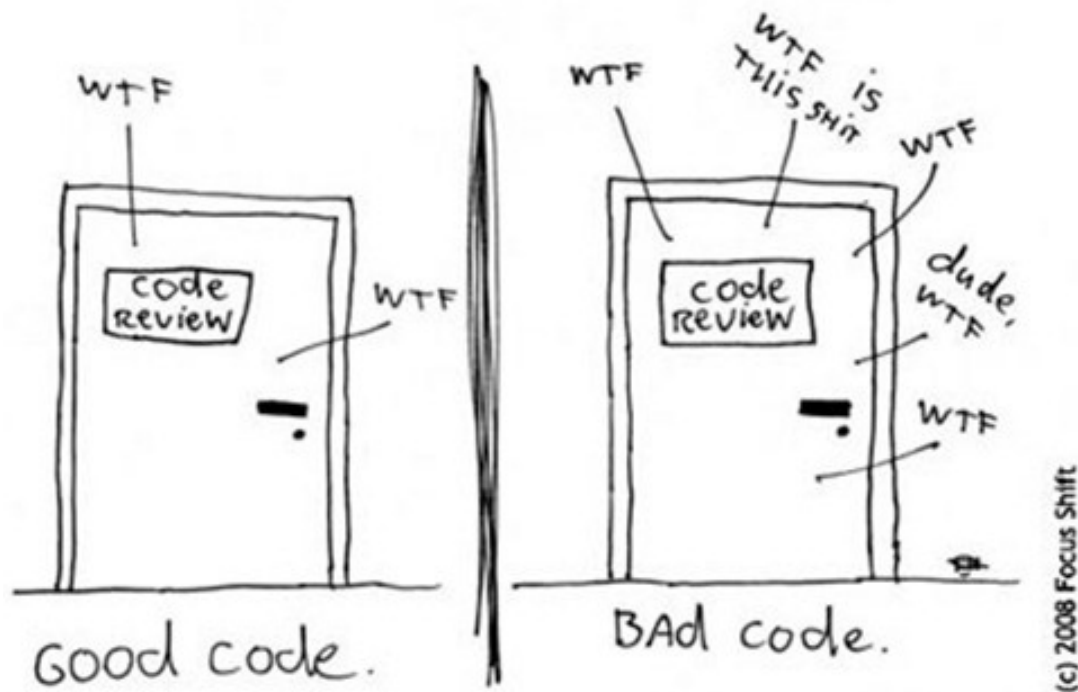
- ma na celu wykrycie i poprawienie błędów w kodzie
- lub zwiększenie jakości poprawnie działającego kodu
- pozwala innym członkom zespołu zapoznać się ze zmianami w projekcie
- znacząca zwiększa jakość
- pułapka “formalnego review”

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





Clean code

Code review

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

Your recently pushed branches:


 develop (less than a minute ago)


 Compare & pull request

- 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

 base: master ... compare: develop ✓ Able to merge. These branches can be automatically merged.



nazwa commita


Write

Preview


AA ▾ B i “ <> 🔗 ⋮ ⋮ ⋮ ↶ @ 📌

Bardziej szczegółowy opis, jeżeli zadanie tego wymaga.
Tutaj można zamieścić link do zadania w JIRA


Attach files by dragging & dropping, [selecting them](#), or pasting from the clipboard.


 Styling with Markdown is supported


Create pull request

Reviewers 


No reviews—request one

Assignees 


 tomliśow

Labels 

None yet

Projects 

None yet

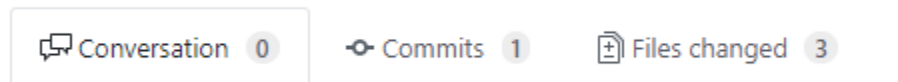
Milestone 

No milestone

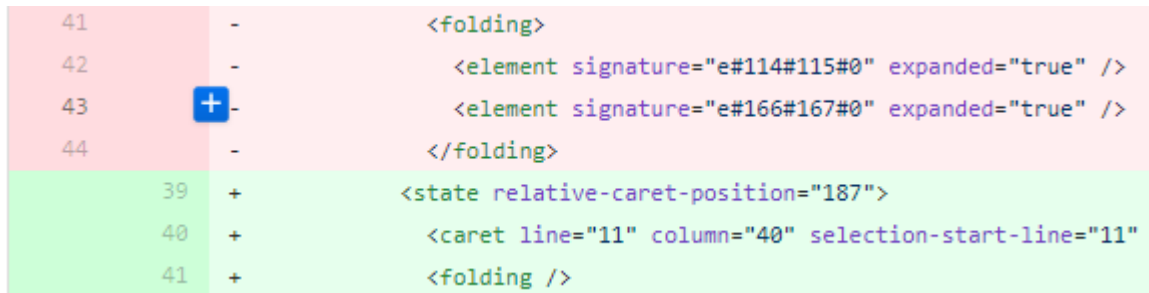
Clean code

Code review

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



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



Clean code

Code review

Unified Split Review changes 1 ▾

Submit your 1 pending comment

Review summary

Wymaga zmiana.

☒ **Comment**
Submit general feedback without explicit approval.

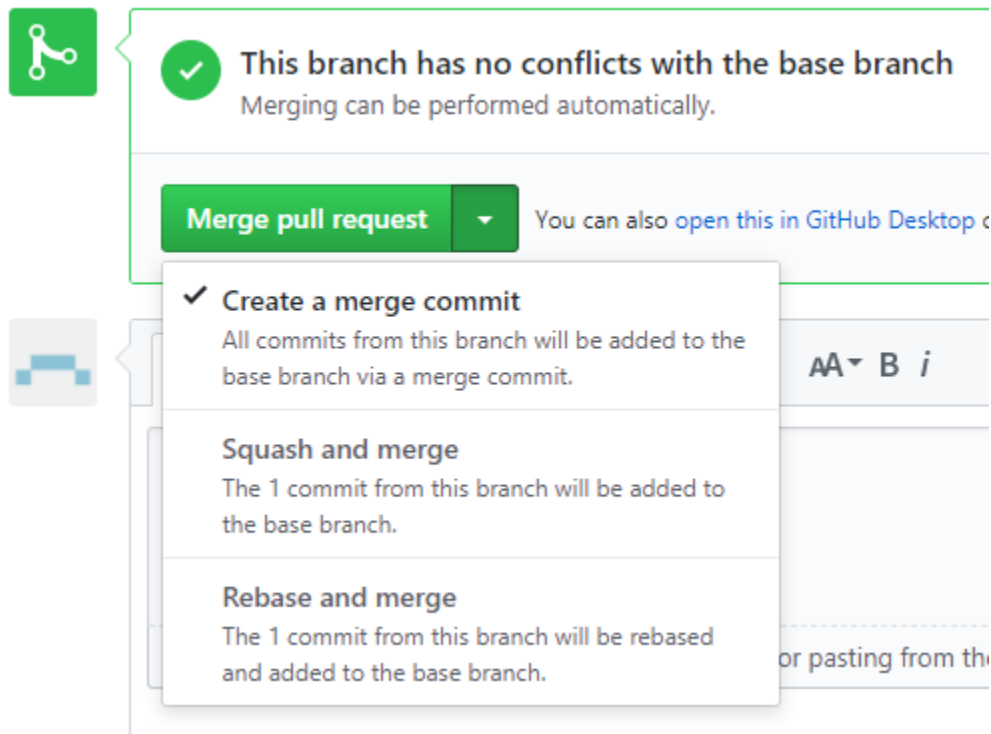
☐ **Approve**
Submit feedback and approve merging these changes.

☐ **Request changes**
Submit feedback that must be addressed before merging.

Submit review

Clean code

Code review



The screenshot displays a GitHub pull request interface. At the top, a green checkmark icon is followed by a green circle containing a white checkmark, indicating that the branch has no conflicts with the base branch. Below this, a green button labeled "Merge pull request" is visible, along with a link to "open this in GitHub Desktop". A dropdown menu is open, showing three options: "Create a merge commit", "Squash and merge", and "Rebase and merge". Each option has a brief description of how the commit will be added to the base branch.

✓ This branch has no conflicts with the base branch
Merging can be performed automatically.

Merge pull request You can also [open this in GitHub Desktop](#)

✓ Create a merge commit
All commits from this branch will be added to the base branch via a merge commit.

Squash and merge
The 1 commit from this branch will be added to the base branch.

Rebase and merge
The 1 commit from this branch will be rebased and added to the base branch.

Clean code

Code review

push to develop for request (#1)

Akceptuję zmiany.

Confirm squash and merge Cancel

Clean code

Pair programming

- programowanie w parach
- jedna stacja robocza
- jedna osoba pisze, druga mówi co 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

tomasz.lisowski@protonmail.ch