Clean code, code review & refactoring

1. Po co robić code review?

> ***Code review*** *– praktyka w dziedzinie inżynierii oprogramowania mająca na celu wykrycie i poprawienie błędów popełnionych w kodzie w czasie fazy pisania oprogramowania, a co za tym idzie, poprawienie jakości tworzonego produktu.*

Czym zmierzyć jakość?

... jakość oprogramowania?

Jakie elementy składają się na jakość oprogramowania?

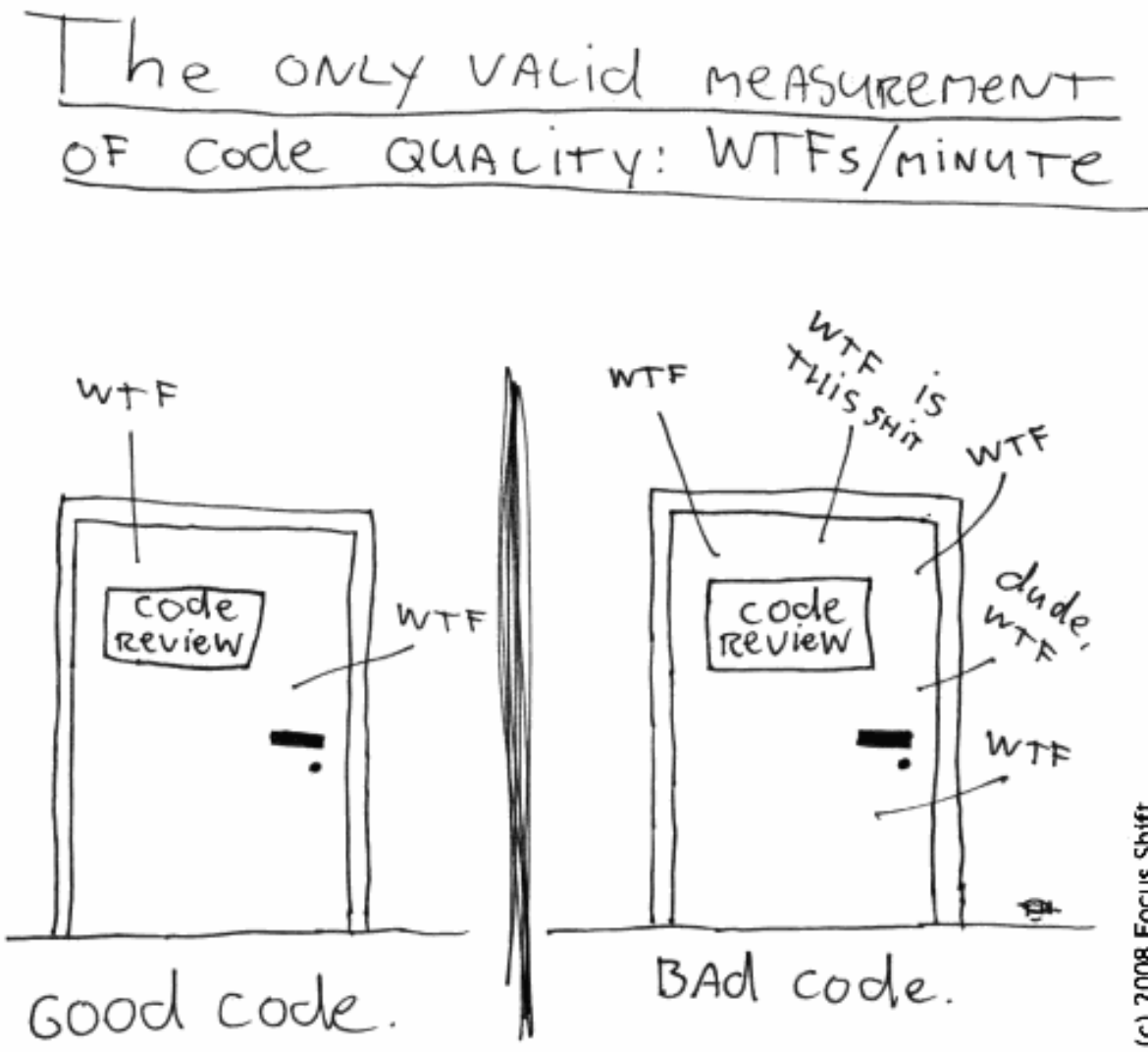
Perspektywa: użytkownika, sponsora, realizatora.

Jak zmierzyć
jakość oprogramowania

- Czy działa
- Czy działa zgodnie z oczekiwaniami
- Ile czasu trwa wprowadzanie zmian
- Czy kod jest zrozumiały dla innej osoby
- Jak szybko działa
- Ile posiada znanych błędów

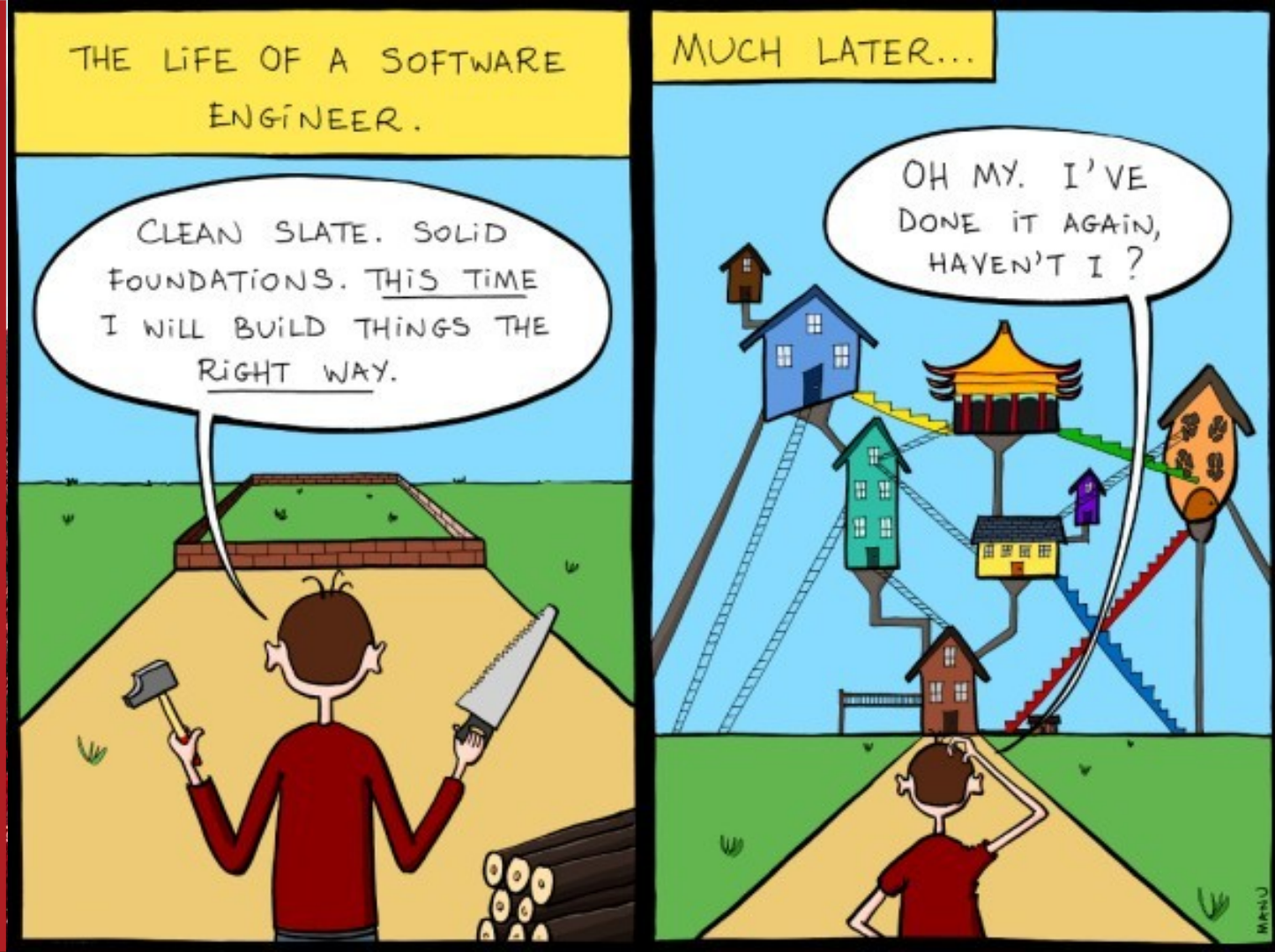
Easy to understand
&
Easy to maintain
&
Best practices

Miara jakości oprogramowania



> *Potrzebuję tu jeszcze jednego małego przycisku. Na kiedy możesz go dodać?*

Użytkownik


THE LIFE OF A SOFTWARE ENGINEER.
CLEAN SLATE. SOLID FOUNDATIONS. THIS TIME I WILL BUILD THINGS THE RIGHT WAY.
MUCH LATER...
OH MY. I'VE DONE IT AGAIN, HAVEN'T I?
MANU

Historia pewnego fuckupa

2. Jak robić code review?

Metody realizacji code review

- Formalne inspekcje (np. Inspekcja Fagana)
- Bezpośredni feedback
- Programowanie w parach
- Z użyciem dedykowanych narzędzi (np. pull request)

Metody realizacji code review
Formalne inspekcje (np. Inspekcja Fagana)

- Korzyści
  - Dobrze opisana
  - Jednoznaczny proces
  - Mierzalna
- Wady
  - „Ciężka”
  - Kosztowna

Metody realizacji code review
Bezpośredni feedback

- Korzyści
  - Realizowane na bieżąco
  - Łatwość przeprowadzenia
  - Brak barier w komunikacji
  - Łatwiejsze zrozumienie tematu przez recenzenta
  - Efektywność
- Wady
  - Presja czasu
  - Wybiórczość
  - Możliwość pominięcia niektórych uwag

Metody realizacji code review
Programowanie w parach

- Korzyści
  - Realizowane na bieżąco
  - Efektywne wykrywanie błędów
  - Przepływ wiedzy w zespole
  - Część programistów lubi taki styl pracy
- Wady
  - 2 razy więcej czasu
  - Brak podziału autor/recenzent
  - Różni programiści w parze
  - Część programistów nie lubi takiego stylu pracy

Metody realizacji code review
Z użyciem dedykowanych narzędzi

- Korzyści
  - Stała praktyka zespołu realizowana nawykowo
  - Forma pisemna uwag
  - Przepływ wiedzy w zespole
  - Transparentność
  - Brak konieczności zgrania czasowego autor/recenzent
- Wady
  - Możliwość przeciągania się w czasie
  - Ping-pong
  - Forma pisemna uwag

Metody realizacji code review
Bezpośredni feedback

- nie bądź ignorantem

Metody realizacji code review
Bezpośredni feedback

- nie wywyższaj się

Metody realizacji code review
Bezpośredni feedback

- nie formułuj bezsensownych żądań

Metody realizacji code review
Bezpośredni feedback

- szanuj partnerów

Metody realizacji code review

GitHub Pull Request

- komentarze do istniejących commitów
- całe commity
- pojedyncze linie
- dyskusje nad pull requestami

“Zawsze pisz kod tak, jakby gość, który ma się nim zajmować był agresywnym psychopatą, który wie, gdzie mieszkasz

Martin Golding

Jak robić dobre code review?

- Zebrać dziesiątki feedbacków na review własnego kodu
- Czytać dobry kod
- Pisać dobry kod
- Robić code review innym (praktyka czyni mistrza)
- Trochę być pedantycznym, ale jednocześnie racjonalnym

Korzyści z code review

- Wczesne eliminowanie błędów
- Redukcja liczby błędów zgłaszanych przez klientów
- Identyfikowanie problemów w procesie

Korzyści z code review

- Rozwija piszącego kod
- Rozwija robiącego review
- Pomaga utrzymać dobre praktyki w projekcie
- Pomaga utrzymać spójny styl kodowania w projekcie
- Każdy fragment kodu oglądany przez 2 pary oczu

Clean code

Good practices HTML + CSS

http://codeguide.co/
https://github.com/airbnb/css

Good practices HTML
W pigułce

- Formatowanie i organizacja kodu
- Podział odpowiedzialności (inline style, inline scripts)
- Tagi HTML5 (h1, section, main, header)
- Zbędny markup
- Style CSS (w head)
- Skrypty JS (na końcu dokumentu)

- Web Accessibility, WAI-ARIA

Good practices CSS
W pigułce

- Formatowanie i organizacja kodu
- Selektory (stylowanie po klasach, unikanie po id, po nazwach tagów html)
- Zagnieżdżenia, kaskadowość
- !important
- Shorthand properties
- Nazewnictwo klas

- Media queries, mobile first
- BEM, 7-1 pattern, ITCSS, OOCSS
- transform: all

Good practices JavaScript

https://github.com/airbnb/javascript

Good practices JS
W pigułce

- Formatowanie i organizacja kodu
- Moduły
- Ograniczenie zmiennych globalnych
- Const, let (unikanie var)
- Nazewnictwo
- Mutowanie danych, map, filter, reduce

- Build tools
- JS lint
- Testy (unit tests)

“ *Nie koduj, pisz prozę!*

Sławek Sobótka

ważne skróty: DRY

- do not repeat yourself

```
if (b != 0) {
    x = a / b;
}
if (d != 0) {
    y = c / d;
}
```

ważne skróty: DRY

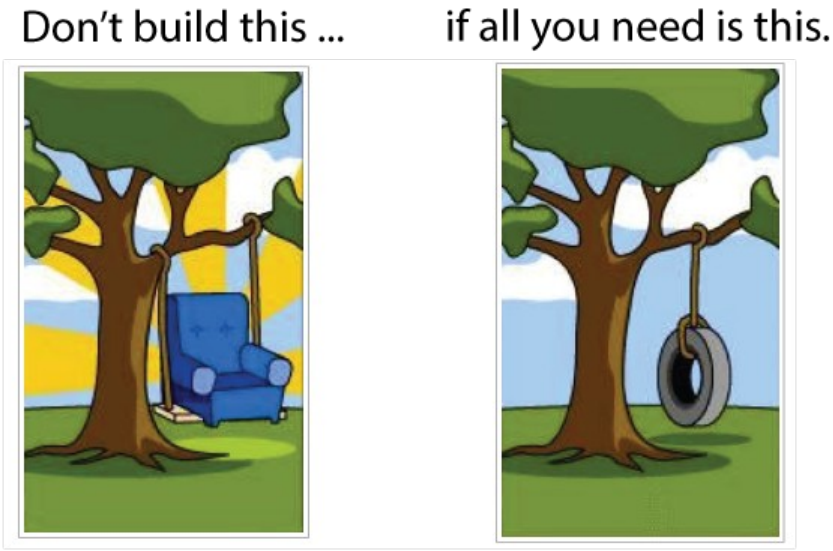
- do not repeat yourself

```
var x = divideNumbers(a, b);
var y = divideNumbers(c, d);

function divideNumbers(dividend, divisor) {
    if (divisor == 0) {
        throw new InvalidArgumentException("Division by zero!");
    }
    return dividend / divisor;
}
```

ważne skróty: YAGNI

- you aren't gonna need it



ważne skróty: YAGNI

- you aren't gonna need it
  - nie zapełniaj spiżarni
  - nie przewidzisz przyszłości
  - skup się tym, o czym wiesz

ważne skróty: KISS

- keep it simple stupid

Refactoring

Refactoring
Definicja

Proces wprowadzania zmian w projekcie/programie, w wyniku których zasadniczo **nie zmienia się funkcjonalność.**
Celem refaktoryzacji jest więc nie wytwarzanie nowej funkcjonalności, ale utrzymywanie odpowiedniej, wysokiej jakości organizacji systemu.

Refactoring
Why?

Programy które

- Są trudne w czytaniu
- Mają powieloną logikę (DRY)
- Zmiany powodują „side effects” i prowadzą do bugów
- Proste problemy są zbyt złożone

Są trudne w rozbudowie!

Refactoring
Pros and cons

Pros:

- Prostszy kod
- Czytelniejszy, łatwiejszy w modyfikacji
- Unikanie potencjalnych błędów

Cons:

- Wymaga czasu (nie zawsze)

Refactoring
Co zrobić gdy nie ma czasu?

Korzystaj ze specjalnych komentarzy

FIXME – powinno być poprawione
HACK – hack :)
TODO – do zrobienia
XXX – uwaga, coś niebezpiecznego

Refactoring
W Czym nie jest refactoring?

- Łatanie bugów
- Optymalizacje wydajności (nie zawsze)
- Defensywny kod
- Pokrywanie kodu testami