# Wybrane zasady z ksiażki „Clean code” R. C. Martin’a

Po co potrzebne?

* *„Kod jest wielokrotnie częściej czytany niż pisany (10:1)”*
* *„Chcemy aby nasz kod dało się szybko zrozumieć a nie intensywanie studiować.”*
* *„Naszym celem jest opowiadanie historii na temat systemu”*
* *„Czytelności jest wszystkim. Profesjonaliści piszą kod zrozumieły dla innych”*
* *„Świeni programiści postrzegają programy jako opowiadania, nie programy do napisania”*
* *„Złożoność zabija”*
* *„Czas poświęcony na wybór odpowiedniej nazwy jest dobrą inwestycją”*

**Znaczące nazwy**

* Powinniśmy używać nazw specyficznych dla domenty oprogramowania.
* Używaj nazw przedstawiających intencje i z dziedziny problemu. Unikaj nazw skrotowych, nieinformujacych. Nie przefiksuj nazw klas jednakowa, żeby potem nie przegladac ich dlugich list.
* Nie należy obawiać się długich opisowych nazw (są lepsze niż krótkie i enigmatyczne)
* Nazwy powinny być łatwe do wyszukania.
* Nazwy klas: Rzeczowniki. Bez słów jak Manager, Processor, Data, Info itp.
* Nazwy metod: Czasowniki.
* Spójny leksykon słów jest ogromnym ułatwieniem dla programistów czytających kod. (nie powinniśmy w kodzie operować zamiennie np.pdf\_id/ens\_id\_guid lub docId/ModuleName itp)
* Nie używaj w naszym kodzie nazw które są stworzone przez systemy zewnętrzne i jednocześnie mylące/dezinformujace. (tutaj sprawdzają się ACL i własne nazewnictwo domenowe).

**Brak powtórzeń (DRY – Don’t repeat yourself)**

* Eliminujemy wszystkie powtórzenia przez ich wydzielenie do jednego miejsca.
* Nie tworzymy kodu ciągle korzystając z ctrl\_c + ctrl\_v (tzw. metody Kopiego Pasty).

**KISS**

* Keep it simple stupid. Kod powinien być tak prosty jak to tylko możliwe.

**Interfejsy**

* Nie dodajemy prefixu „I”. W nazwach implementajci możemy dodawać postfix „Impl”
* Dobrze zdefiniowane moduły mają bardzo małe interfejsy. Źle zdefiniowane moduły mają szerokie i głębokie interfejsy, które wymymuszą przesyłania dużych konstrukcji do wykonania prostych operacji.

**Argumenty**

* Idealną liczbą do argumentów funkcji jest 0.
* Im mniej argumentów tym lepiej. Nie powinno być więcej niż 3.
* Obiekty przekazywane przez agrumenty do metody nie powinny być w niej modyfikowane.
* Metoda powinna zwracać przez return modyfikowany obiekt. Lub modyfikacja być może powinna być częścią danego obiektu.
* Argumenty wyjściowe – nie stosować, mało intuicyjne.
* Jeśli metoda ma zmieniać stan czegokolwiek to powinna zmieniać stan tylko własnego obiektu.

**Metody**

* Metody nie powinny mieć efektów ubocznych. Powinny wykonywać jedną operację.
* Funkcje powinny być małe, tak aby mieściła się na ekranie monitora (a nawet do 20 linii).
* Funkcje powinny robić jedną operację. Powinnismy mieć oddzielone funcje zmieniające stan modelu od tylko czytających.
* Powinny mieć jeden poziom abstrakcji. W kodzie o wysokim poziomie abstrakcji nie powinniśmy mięć kodu o niskim np append(„\n”)

**Wyjątki**

* Kontrolowane vs niekontrolowane.
* Powinniśmy preferować wyjątki niekontrolowane(runtime), tak aby nie musieć wszędzie w kodzie ich wyłapywać, tylko na najwyższym poizomie gdzie to naprawdę potrzebne. C++/Python/Ruby nie mają zupełnie wyjatków kontrolowanych.
* Kod optymistycznego flow powinien być rozdzielony od kodu obsługi wyjątku.
* Nie powinniśmy tworzyć wiele klas wyjątków jeśli obsługa jest jednakowa.

**Pola**

* Za duża liczba pól w klasie sugeruje potrzebę nowej klasy.

**Komentarze w kodzie**

* *„Komentarze są w najlepszym przypadku koniecznym złem.”*
* *„Za każdym razem gdy piszemy komentarz powinniśmy poczuć smak porażki”*
* *„Czytelny kod nie wymaga komentarzy.”*
* Unikać nadmariowych komentarzy. Każdy komentarz = potencjalnie coś jest nie tak z kodem, że jest nieczytelny. Nie opisywać co robi kod tylko jakie są intencje stworzenia tego kodu.
* Gdy piszę komentarz to zawsze powinienem się zastanowić się dlaczego kod nie mówi jasno jak robi?

**Różne**

* Program powinien mieć jednorodny styl (nie połączenie wielu styli indiwidualnych developerów)
* Nie chcemy udostępniać szczegółów danych naszej klasy, najgorsza opcja to automatyczne wygenerowanie dla klasy seterów/getterów (zwłaszcza klasy bizesowej). Moduł nie powinien wiedziec nic o wnętrzu obiektów którymi manipuluje.
* Obiekty udostępniają operacje i ukrywają dane. Struktury danych udostępniają dane i nie mają znaczących operacji.
* Nie zwracamy wartości NULL, aby wołające funkcje za kazdym razem nie musiały tego kontrolować.
* Nie uzalezniać się od zewnętrznego kodu.
* W klasie umieszczamy w kolejności o jej góry: pola publiczne, pola prywatne, metody publiczne, metody prywatne.
* Im mniej metod ma klasa / im mniej zmiennych jest wykorzystywanych w funkcji / im im mniej zmiennych instancyjnych ma klasa ....tym lepiej.
* Liczby magiczne powinny być w kodzie używane przez dobrze nazwane zmienne.
* Warunki pozytywne (if sth) są czytelniejsze niż warunki negatywne (if !notSth).
* Stosować kompozycję przed dziedziczeniem.
* Czytając czysty kod nie będziemy zaskakiwania (mała liczna wft/min)
* Minimalizacja liczby encji (klas,metod,funkcji,itp)