



Refaktoring

Sztuka Wytwarzania Oprogramowania, w. 13

Konrad Grochowski

Instytut Informatyki, Politechnika Warszawska, 2025 ©





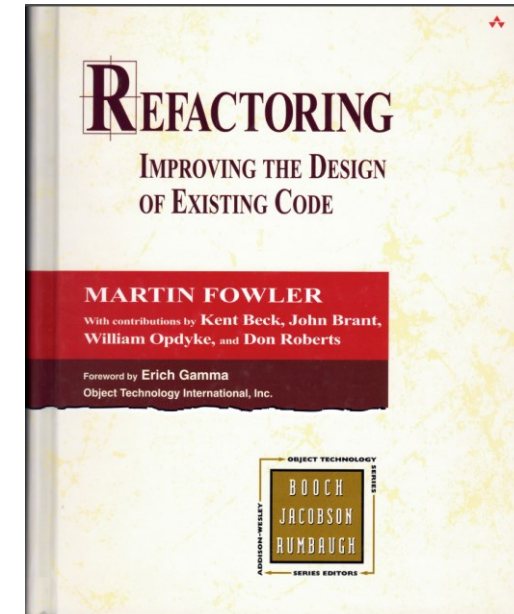
Refaktoring

- › Modyfikacja projektu aplikacji bez zmiany jej funkcjonalności
- › Cel: poprawa jakości i utrzymywalności projektu (kodu)
- › Kiedy: (*prawie*) kiedy tylko się da
 - W TDD to obowiązkowy krok w każdym mini-cyklu
 - W dużych/starych projektach okazuje się często niezbędny by dodać obsługę nowego wymagania
 - Albo po prostu – widzę zły kod – poprawiam (o ile mam czas etc.)



Refaktoring

- › Dość stare pojęcie
 - Martin Fowler – Refactoring (1999)
- › Książka *bardzo* warta przeczytania
 - Druga edycja – 2018 (*JavaScript...*)
 - Ale nawet pierwsza edycja pokazuje, że pewne problemy są „ponad czasowe” i nie „wymyślone wczoraj”





Dlaczego refaktoryzować kod?

- › Dług techniczny (*technical debt*) projektów rośnie
 - i kapitalizuje odsetki
- › Nikt nie jest idealny
- › Kod *gnije*:
 - Nowe wymagania mogą być niewygodne do implementacji
 - Zmiany w *innych* częściach systemu mogą wpłynąć na to jak dany fragment „pasuje” do reszty (nieefektywny, nazbyt skomplikowany etc.)
 - Wiedza zespołu o domenie się powiększa w trakcie tworzenia projektu i pewne rzeczy później można zrobić *lepiej*
 - Zmiany technologiczne, nowe biblioteki, czy nawet *moda*



Najgroźniejszy błąd refaktoringu

- › „*To jest takie złe, weźmy to wszystko napiszmy od zera*”
- › To nie jest refaktoring
- › To (*najczęściej*) strzelanie w stopę sobie
 - i projektowi
 - i budżetowi
- › Z haubicy



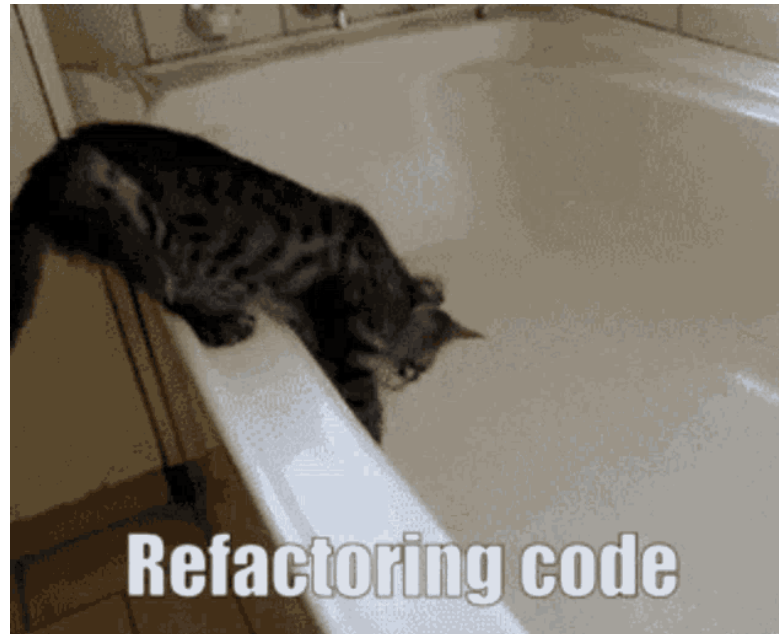
Druga skrajność - strach

- › „Jak działa to nie ruszać”
 - Dobre hasło, ale zależy od definicji *działa*
 - *Utrzymywalność* często definiuje, czy *biznes* działa (a nie tylko kod)
- › Strach przed regresją oprogramowania (utrata czegoś)
- › Jedyne co pomaga na taki strach – testy
 - Pokrywające całość (znaczącą większość) oczekiwań klienta
 - Albo przynajmniej tę część co chcemy zrefaktoriować



Druga skrajność - strach

- › Strach przed „ugrzeźnięciem” w refaktoringu



- › Tylko *profesjonalizm* i *spokój* mogą wtedy uratować :)



Dobry refaktoring

- › Robiony *małymi krokami*
- › Testowany na każdym kroku – czy funkcjonalność nie jest zmieniana
- › Spowodowany możliwie obiektywnymi powodami
- › (Zazwyczaj) robiony „przy okazji” jakiegoś innego zadania
- › Zakańczany w *rozsądnym* czasie
 - *Problem*: programiści *lubią* refaktoryzować (jak już przełamią strach)
 - Nie ma dobrej reguły „kiedy kończyć”
 - Najczęściej jest to skrzyżowanie „wyczucia”, budżetu/planowania czasu i opinii zespołu



Po czym poznać, że kod wymaga refaktoringu?

- › Zgniły kod posiada *zapachy (code smells)*
- › Przykłady:
 - Zduplikowany kod
 - Nieczytelne nazwy
 - Długa metoda
 - Duża klasa
 - Dużo parametrów
 - Parametry boolowskie
 - Złamane reguły SOLID
 - Zazdrość (*feature envy*)
 - Zlepki danych (*data clumps*)
 - Obsesja prymitywów
 - Wyrażenie *switch*
 - Złożony schemat dziedziczenia
 - Nieuzasadniona generyczność
 - Tymczasowa wartość
 - Wywołania łańcuchowe
 - Komentarze



Operacje refaktoryzujące (przykłady)

- › *Extract Method*
- › *Inline Method*
- › *Replace Temp with Query*
- › *Move Method / Field*
- › *Extract / Inline Class*
- › *Remove Middle Man*
- › *Encapsulate Field*
- › *Rename Method / Field*
- › *Collapse Hierarchy*
- › *Pull Up Field / Method*
- › *Push Down Field / Method*
- › *Extract Interface / Superclass*
- › *Separate Query From Modifier*
- › *Replace Parameter with Explicite Methods*
- › *Replace Inheritance with Delegation*



Jak refaktorować?

- › Trzeba mieć jakieś testy
 - Jak w projekcie ich nie ma, trzeba zacząć od ich dodania
 - Szybkie testy jednostkowe są najwygodniejsze do refaktoringu
- › Małe kroczki, jak najczęściej weryfikowane
 - W oryginale „*Extract Method*” było rozpisane na *dziwięć (9)* kroków
 - A teraz często robi to za nas IDE, ale to dobry przykład pokazujący, jak coś potencjalnie prostego, może zawierać wiele „haczyków”.



A jak pracować z „zastanym kodem” (*legacy code*)?

- › Nie bać się refaktoriować
 - Ale pamiętać o testach
- › Nie robić od razu rewolucji
 - Ewolucja i małe zmiany są bezpieczniejsze
- › Nie naruszać lokalnego standardu kodowania
 - O ile nie jest bardzo przestarzały i nie wymusza bardzo złych praktyk
- › Przede wszystkim – realizować bieżące potrzeby klienta
 - Jeśli okaże się, że kod jest w takim stanie, że bez *planowego* refaktoringu prace będą szły coraz wolniej – udowodnić to, omówić z „kierownictwem” i wprowadzić *plan refaktoringu*



Refactoring Katas

- › <https://github.com/sandromancuso/trip-service-kata>
- › <https://github.com/jbrains/trivia>
- › <https://github.com/emilybache/Tennis-Refactoring-Kata>
- › <https://github.com/emilybache/GildedRose-Refactoring-Kata>

Dziękuję za uwagę

Konrad.Grochowski@pw.edu.pl

