



Projektowanie i programowanie

Sztuka Wytwarzania Oprogramowania, w. 5

Konrad Grochowski

Instytut Informatyki, Politechnika Warszawska, 2023 ©





Ale najpierw – c.d. SOLID, Coding Standards etc.

- › Wszystkie opisane wcześniej reguły są *subiektywne*
- › Pomagają poprawiać *utrzymywalność* kodu, czyli potencjał na dostarczanie szybko poprawnie działających nowych wersji
- › Ale (zazwyczaj) nie odnoszą się do samej *poprawności* działania programu
- › A co pozwala weryfikować poprawność?



Testowanie w wytwarzaniu oprogramowania

- › **Bez testów nie ma programu**
- › Jest więcej rodzajów testów, niż tylko jednostkowe...
- › ...ale one są najlepszym przyjacielem programisty



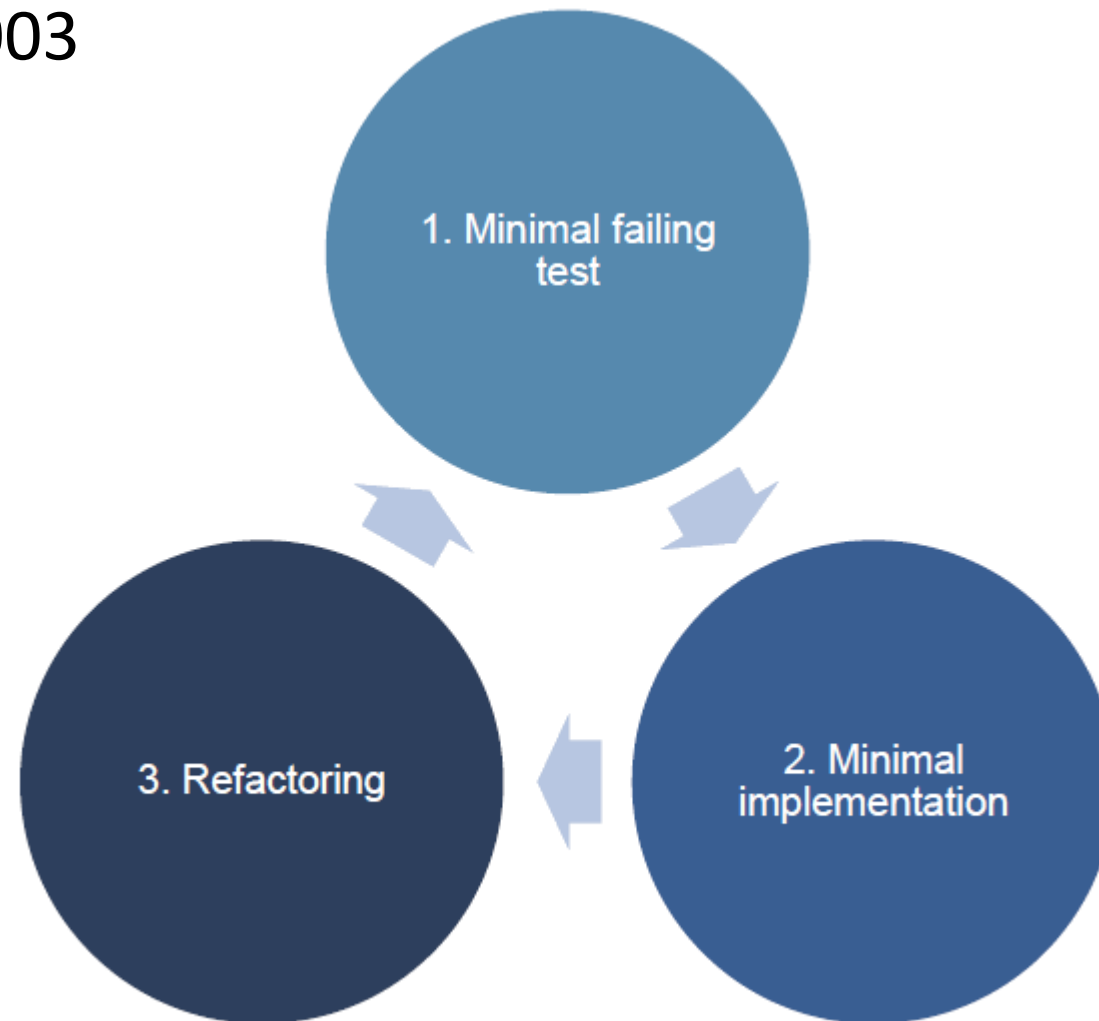
Test-Driven Development

- › Pomysł – używać testów nie tylko do weryfikacji procesu wytwarzania oprogramowania, ale do *sterowania* nim
- › To *nie* jest „najpierw napiszę testy a potem cały kod”
- › Bliższe prawdy zdanie: „pozwolę testom się poprowadzić przez rozwiązanie problemu”.
- › Ważne – TDD nie zastępuje testowania oprogramowania – to inne procesy!
- › Im kto gorzej sobie radzi z testami i tworzeniem kodu, tym więcej da mu stosowanie się do reguł/rygorów TDD



Test-Driven Development

- › Kent Beck 2003
- › 3-step cycle:





Three Rules of TDD

- › Write production code only to pass a failing unit test.
- › Write **no more** of a unit test than sufficient to fail
(compilation failures are failures).
- › Write **no more** production code than necessary to pass the **one** failing unit test.



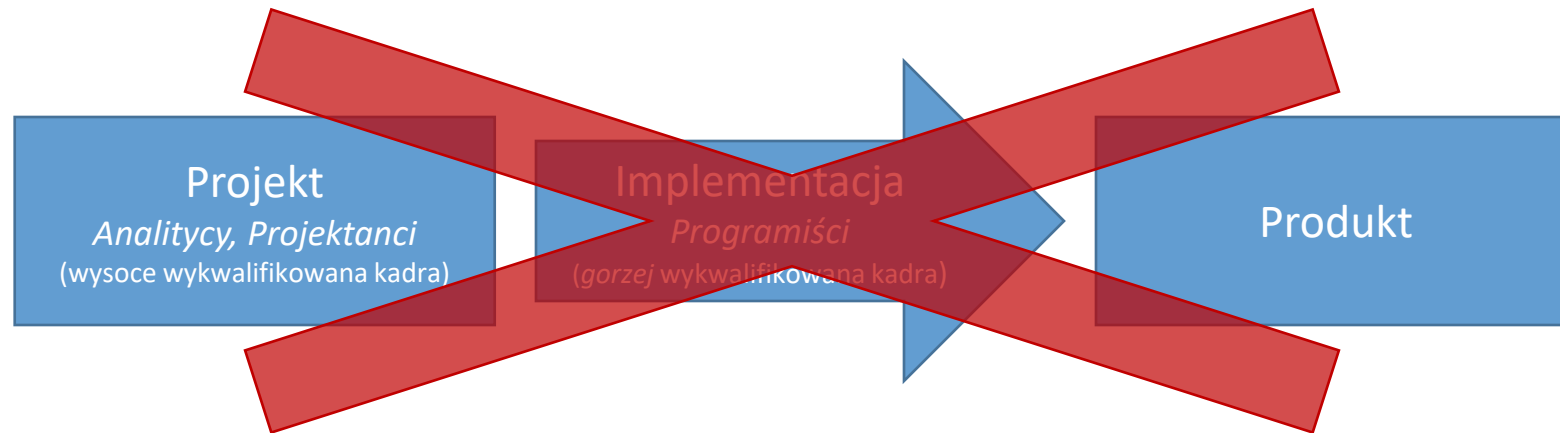
Live Demo

- › Prymitywny przykład, ale pozwala zrozumieć „sterowanie”



Co to jest projekt?

- › Niekiedy rozumiany jako „wysokopoziomowa architektura rozwiązania”
- › Co niektórych prowadzi do takiego procesu:



- › Jest to *bardzo* błędne rozumienie procesu inżynieryjnego
- › A w szczególności efekt zmiksowania niepoprawnych wizji procesu budowlanego i linii produkcyjnej



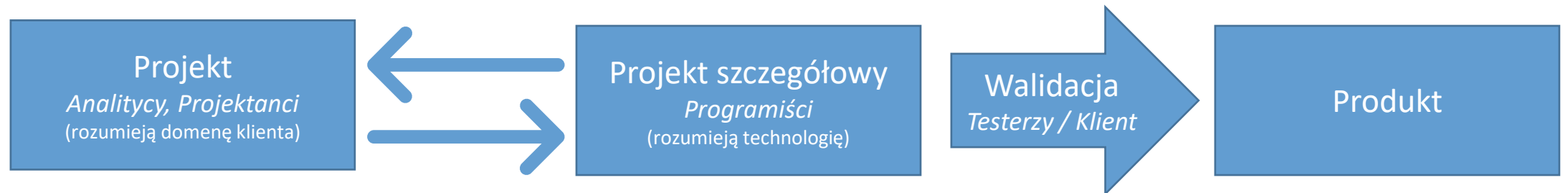
Projekt

- › Projekt to wszystko *po drodze* do rzeczywistego produktu
- › Kiedy aplikacja staje się rzeczywistością?
- › Wtedy gdy zabiera się do jej stworzenia kompilator/interpreter
- › Czyli – *kod jest projektem aplikacji*
- › Poprawniej – kod **też** jest częścią projektu aplikacji
- › O **kluczowym** wpływie na to, czym aplikacja będzie, ale częścią
- › Jak ktoś potrzebuje analogii – *projekt wykonawczy*,
a może nawet *projekt **powykonawczy***



Proces inżynieryjny (zarys)

- › To bardzo ogólny zarys, będą o tym kolejne wykłady
- › I całe przedmioty, bo dziedzina jest ogromna i bardzo ważna
- › *Inżynieria Oprogramowania / Software Engineering*



- › Dwukierunkowość można osiągnąć np. współdzieląc osoby
- › *Obowiązkiem* inżyniera, jest pilnować jakości procesu
(*zmieniać firmę*)



Projekt i kod

- › Kod nie jest idealny do łatwego demonstrowania architektury
- › (Ale ona *musi* być w kodzie – jak się nie zgadzają, to architektura może iść do kosza, bo to kod efektywnie „idzie do klienta”)
- › Rysunek bywa najefektywniejszą formą przekazu informacji
- › Ale trzeba pamiętać, że w razie wątpliwości – *sprawdzać kod*
- › Zabiegi pomocnicze:
 - Nazewnictwo oparte domenę i architekturę
 - Struktura pakietów/folderów/plików zgodna z architekturą
 - *Rozsądna* dokumentacja kodu



Rysowanie a programowanie

- › Diagram, który może zostać aplikacją, musi być jednoznaczny, kompletny i konkretny
- › Mamy wtedy do czynienia z *graficznym językiem **programowania***
- › Czyli obowiązują wszystkie reguły „tworzenia kodu” i podobny sposób myślenia
 - Szczegółowe diagramy mają sens tylko jeśli stają się „plikami źródłowymi”
- › Marzenia o „rysowaniu programów przez nie-programistów” są raczej nierealne
 - A przynajmniej *nie każdego* programu

Dziękuję za uwagę

Konrad.Grochowski@pw.edu.pl

