



# Projektowanie i programowanie

*Sztuka Wytwarzania Oprogramowania, w. 3*

Konrad Grochowski

Instytut Informatyki, Politechnika Warszawska, 2024 ©





## 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



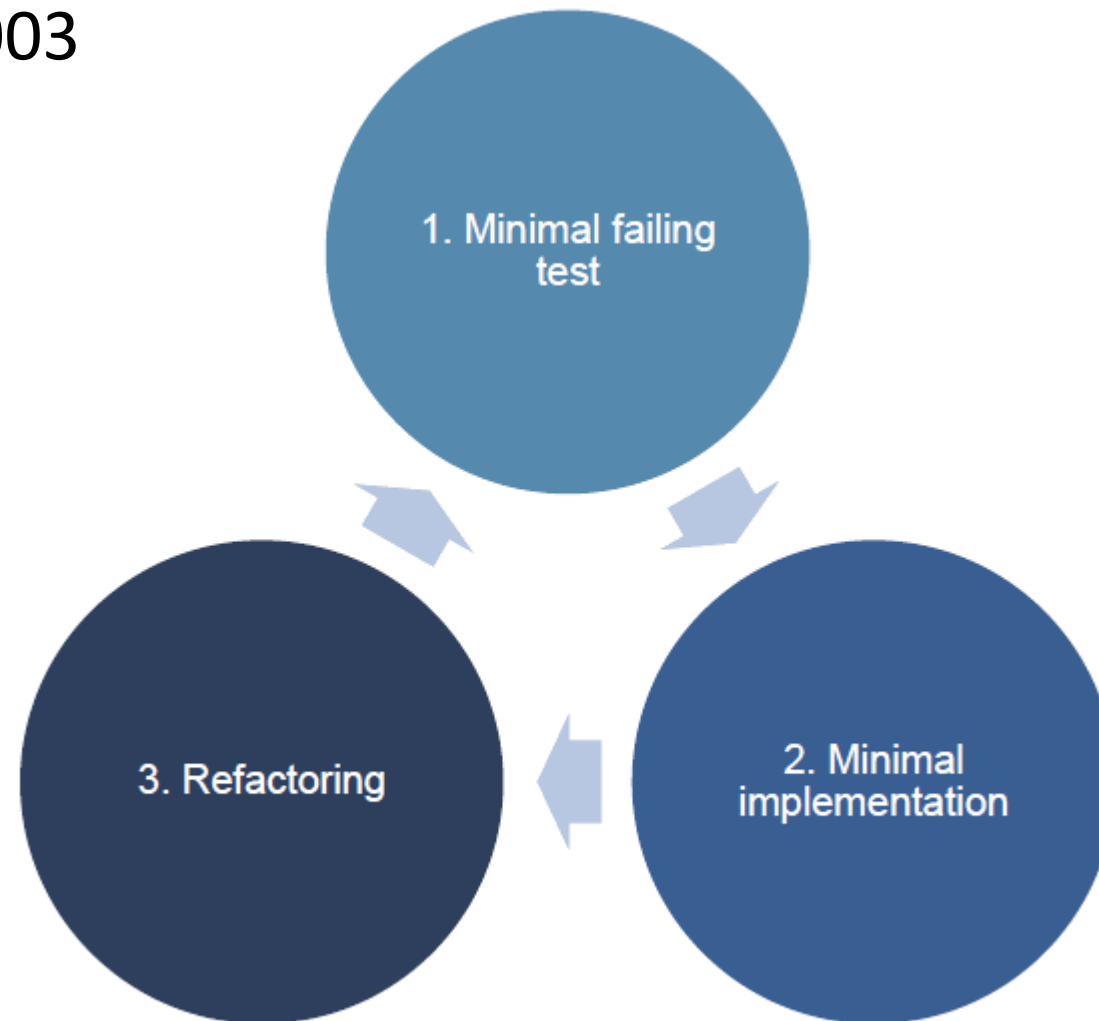
# Test-Driven Development

- › 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”



## Dlaczego warto spróbować

- › Większa pewność (siebie i kodu)
- › Mniejszy *strach przed zmianą (Fear of Change)* – mniejszy stres
- › Lepsza koncentracja
- › Poczucie osiągnięć (*achievement*) i satysfakcji z pracy
  
- › To są argumenty *psychologiczne* ale inżynier nie może zapominać, że jest człowiekiem

# Dziękuję za uwagę

[Konrad.Grochowski@pw.edu.pl](mailto:Konrad.Grochowski@pw.edu.pl)

