rule FINANCIAL

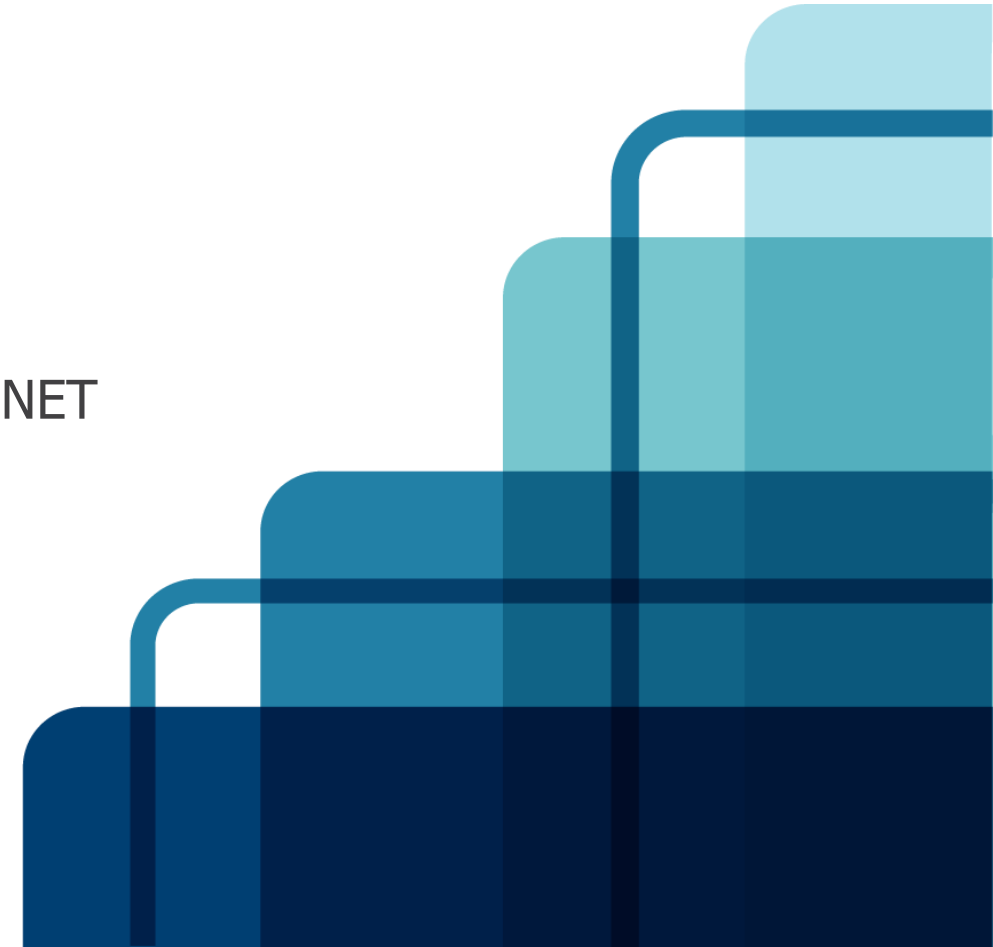
The Sector Specialists

Unit Tests in real life

projektowanie rozproszonych aplikacji w .NET

Prepared by: Janek Polak

Version: 1.0



Confidential

Dobre praktyki

- Utrzymujmy biblioteki interfejsów **do wszystkiego:** IFileService (I/O na dysk), IInteractionService (okienka z błędami), ILogger (wrapper wokoło NLog/Log4Net).
- Pisanie testowalnego kodu to ważna umiejętność. Klasy powinno się dać testować jednostkowo nawet gdy nie planujemy pisać testów.
- Jeżeli dane z zewnętrznych źródeł są niezbędne można je dołączyć np. jako zserializowane dto do zasobów testów.

Czego unikać

- Pisania testów tylko po to, żeby dobić do 100% pokrycia kodu,
- Przeklejania klasy testowanej w test jednostkowy, żeby porównać wyniki algorytmu z samym sobą,
- Duplikowania testów – jeżeli testy współdzielą dużo kodu może trzeba zmienić klasy testowane?

Testy jednostkowe to nie święty graal programowania

Problemy z testami jednostkowymi

- Możliwość pozostawienia przypadków brzegowych bez pokrycia przy teoretycznym pokryciu kodu 100%,
- Zniechęcają do programowania defensywnego,
- Nie wykrywają problemów integracyjnych
- Źle zaprojektowane mogą być bardzo kosztowne w utrzymaniu
- Łatwo można oszukiwać przy ich pisaniu :D

Naruszanie podstaw testów jednostkowych

- Jeżeli testujemy serwis (bezstanowy) zazwyczaj mamy interfejs zewnętrzny za którym się chowa cała logika (parsowanie wiadomości, routing, przetwarzanie danych, składanie odpowiedzi). Test tej klasy z zestawionym maksymalnie dużym backendem często da tyle samo co testy jednostkowe każdej części z osobna a jest dużo tańszy.
- Przy silnych powiązaniach między klasami albo rozbudowanych drzewach wykonania funkcji lepiej pisać testy rozpinające całą ścieżkę, niż tylko jedną metodę.
- Trzeba zachować umiar
- NUnit to bardzo dobre narzędzie do testów integracyjnych z bazą danych.