Politechnika Wrocławska

Platformy programistyczne

Platforma .NET

Dr inż. Radosław Idzikowski

Katedra Automatyki, Mechatroniki i Systemów Sterowania
Wydział Informatyki i Teleinformatyki

9 kwietnia 2024

HR EXCELLENCE IN RESEARCH

Platforma .NET

Platforma .NET

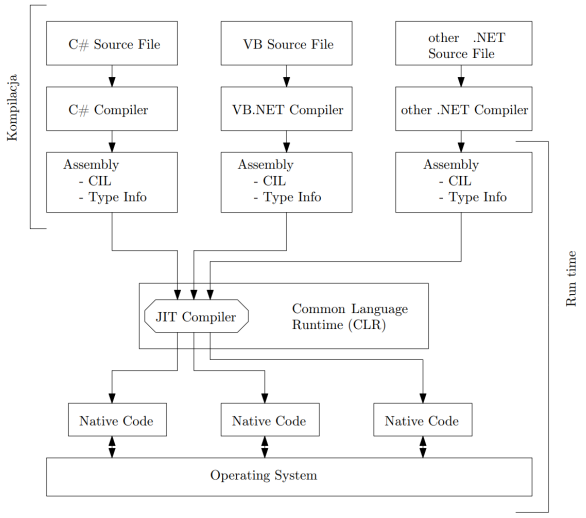
Darmowa platforma programistyczna typu open-source, kompatybilna z wieloma systemami operacyjnymi, przeznaczona do tworzenia różnego rodzaju aplikacji. Można na niej uruchamiać programy napisane w wielu językach, z których najpopularniejszy jest C#. Platforma ta opiera się na wydajnym środowisku wykonawczym, które jest wykorzystywane w produkcji przez wiele skalowalnych aplikacji.

.NET to nie tylko C#

Komponenty

- **CLR** (*Common Language Runtime*) – odpowiada za lokalizowanie, wyczytywanie i zarządzanie typami `.NET`. Ma analgoiczne działanie do maszyny wirtualnej `Java`. Należy do niego zadanie kompilowania i uruchamiania kodu, który jest zapisany językiem kodu pośredniego **CIL** (*Common Intermediate Language*), przez co CLR stanowi trzon całej platformy `.NET`.
- **CTS** (*Common Type System*) – odpowiada za opis wszystkich danych, które są udostępniane przez środowisko uruchomieniowe.
- **CLS** (*Common Language Specification*) – jest zbiorem zasad, które definiują podzbiór wspólnych typów precyzujących zgodność kodu binarnego z dostępnymi kompilatorami `.NET`.
- **BCL** (*Base Class Library*) – zestaw podstawowych bibliotek.

Proces kompilacji .NET

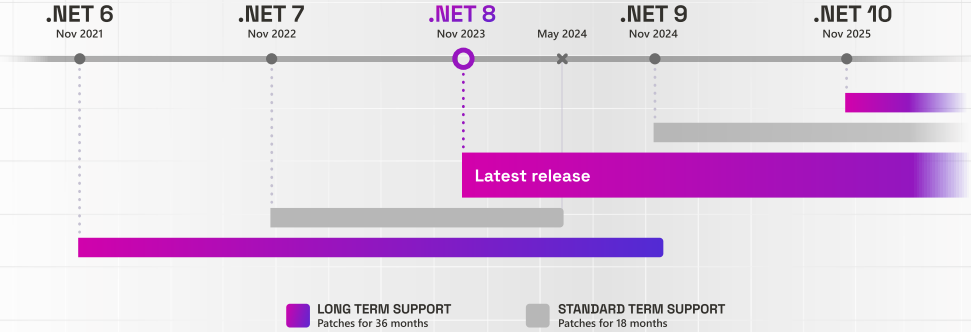


Wersje .NET

- .NET Framework – pierwsza wersja platformy .NET (od 2002),
- .NET dawniej .NET Core – wieloplatfromowa i otwarta wersja platformy .NET rozwijania od 2016 roku, aktualna wersja .NET 8.0.
- .NET Standard to specyfikacja, która określa zestaw interfejsów programowania aplikacji (API), które muszą być dostępne we wszystkich implementacjach platformy .NET, niezależnie od systemu operacyjnego czy architektury.
- Mono – oryginalna społeczność i platforma .NET typu open source rozwijania przez Xamarin. Implementacja międzyplatformowa .NET Framework. Aktywnie obsługiwane w systemach Android oraz iOS.

timeline

Wsparcie .NET



Wersja C# 1.0

2002

- Początkowo przypominała Javę w projektowaniu
- Brak nowoczesnych funkcji, takich jak async i generics
- Brak LINQ
- Pomimo ograniczeń, stanowiła wiarygodną alternatywę dla Javy na platformie Windows

- Główne cechy C# 1.0:
- Klasy
- Struktury
- Interfejsy
- Zdarzenia
- Właściwości
- Delegaty
- Operatory i wyrażenia
- Instrukcje
- Atrybuty

C# w wersji 2.0

2005

- Generics (ogólniki)
- Częściowe typy
- Anonimowe metody
- Nullable value types (typy wartości opcjonalne)
- Iteratory

Inne cechy C# 2.0:

- Oddzielna dostępność dla getterów/setterów
- Konwersje grupy metod (delegaty)
- Statyczne klasy
- Inference dla delegatów

Podczas gdy C# zaczął jako ogólnikowy język obiektowy, wersja 2.0 zmieniła to szybko. Dzięki ogólnikom, typy i metody mogą działać na dowolnym typie, zachowując jednocześnie bezpieczeństwo typów. Wersja C# 2.0 wprowadziła iteratory, które znacząco poprawiły czytelność języka i zdolność ludzi do rozumienia kodu.

C# w wersji 3.0

2007

- Auto-implemented properties
- Anonymous types
- Query expressions
- Lambda expressions
- Extension methods
- Implicitly typed local variables
- Partial methods
- Object and collection initializers

W perspektywie retrospektywnej wiele z tych funkcji wydaje się zarówno nieuniknione, jak i nierozłączne. Wszystkie one strategicznie ze sobą współgrają. Killer feature tej wersji C# to wyrażenie zapytania, znane również jako `Language-Integrated Query` (`LINQ`).

C# w wersji 4.0

2010

Nowe funkcje w C# 4.0:

- Dynamic binding
- Named/optional arguments
- Generic covariant and contravariant
- Embedded interop types

Najważniejszą cechą jest wprowadzenie słowa kluczowego `dynamic`. Wprowadza możliwość nadpisania kompilatora w kwestii typowania w czasie kompilacji. Używając słowa kluczowego `dynamic`, można tworzyć konstrukcje podobne do języków dynamicznie typowanych, takich jak JavaScript.

C# w wersji 5.0

2012

Najważniejsza cecha w C# 5.0:

- Asynchroniczne metody

Wprowadzenie funkcji `async` i `await`, pozwoliło na asynchroniczność wykonywania fragmentów kodu.

C# w wersji 6.0

2015

Główne cechy w C# 6.0:

- Importowanie statyczne
- Filtry wyjątków
- Inicjatory automatycznej właściwości
- Interpolacja łańcuchów
- Operator nameof

Inne nowe funkcje:

- Inicjatory indeksów
- Oczekiwanie w blokach catch/finally
- Domyślne wartości dla właściwości tylko do odczytu

Patrząc na te funkcje razem, zauważamy interesujący wzorzec. W tej wersji C#, zaczęto eliminować zbędny kod językowy, aby sprawić, że kod był bardziej zwięzły i czytelny. Dla miłośników czystego, prostego kodu, ta wersja języka była ogromnym sukcesem.

C# w wersji 7.0

2017

Nowe funkcje w C# 7.0:

- Zmienne out
- Tuple
- Lokalne funkcje
- Rozszerzone ciała wyrażeń
- Ref zmienne lokalne
- Ref zwracane wartości

Wszystkie te funkcje oferują nowe możliwości dla programistów i szansę na pisanie bardziej czytelnego kodu niż kiedykolwiek wcześniej. Ciekawą cechą jest skrócenie deklaracji zmiennych do użycia z słowem kluczowym out oraz możliwość zwracania wielu wartości za pomocą tuple. .NET Core teraz docelowo obsługuje każdy system operacyjny i skupia się na chmurze oraz przenośności.

C# w wersji 8.0

2019

Nowe funkcje w C# 8.0:

- Deklaracje using
- Statyczne lokalne funkcje
- Typy odniesienia nullable
- Strumienie asynchroniczne
- Indeksy i zakresy
- Ulepszenie interpolowanych surowych łańcuchów

Nowe funkcje i ulepszenia w C# 8.0 wykorzystują nowe możliwości Common Language Runtime (CLR) oraz nowe typy biblioteki dodane tylko w .NET Core 3.0.

C# w wersji 9.0

2020

- Rekordy
- Własności tylko do inicjalizacji
- Instrukcje na najwyższym poziomie
- Wskaźniki do funkcji
- Nowe funkcje dla metod częściowych
- Wyrażenia new z określonym typem docelowym
- Wyrażenia warunkowe z określonym typem docelowym
- Wsparcie GetEnumerator dla pętli foreach jako rozszerzenia

C# 9 kontynuuje dwa główne tematy z poprzednich wersji: oddzielanie danych od algorytmów oraz dostarczanie większej liczby wzorców w większej liczbie miejsc.

C# w wersji 10.0

2021

- Usprawnienia w typach strukturalnych
- Globalne dyrektywy using
- Deklaracje przestrzeni nazw na poziomie pliku
- Rozszerzone wzorce właściwości
- Usprawnienia w wyrażeniach lambda
- Dozwolone stałe łańcuchy interpolowane

Globalne dyrektywy using i deklaracje przestrzeni nazw na poziomie pliku oznaczają wyraźniejsze wyrażanie zależności i organizacji przestrzeni nazw. Usprawnienia w wyrażeniach lambda ułatwiają deklarowanie wyrażeń lambda tam, gdzie są używane. Nowe wzorce właściwości i usprawnienia dekonstrukcji tworzą bardziej zwięzły kod.

C# w wersji 11.0

- Wsparcie dla generycznych operacji matematycznych
- Generyczne atrybuty
- Literały łańcuchów UTF-8
- Wzorce list
- Struktury domyślne automatyczne
- Rozszerzony zakres nameof
- Ulepszone przekształcenie grupy metod w delegat

C# w wersji 12.0

2023

- Konstruktory pierwotne – Możesz tworzyć konstruktory pierwotne w dowolnym typie klasy lub struktury.
- Wyrażenia kolekcji – nowy składniowy sposób określania wyrażeń kolekcji, w tym operator rozprzestrzeniania (..), do rozszerzania dowolnej kolekcji.
- Parametry opcjonalne w wyrażeniach lambda - można definiować domyślne wartości dla parametrów w wyrażeniach lambda.
- Parametry ref readonly – umożliwiają większą klarowność dla interfejsów API, które mogą używać parametrów ref lub in.

Ogólnie rzecz biorąc, C# 12 dostarcza nowe funkcje, które sprawiają, że pisanie kodu C# jest bardziej produktywne. Składnia, którą już znasz, jest dostępna w większej liczbie miejsc. Inna składnia zapewnia spójność dla powiązanych koncepcji.

Wybrane technologie

UI – Windows Forms

- najstarsza i jeszcze popularna technologia tworzenia aplikacji typu rich-client w .NET Framework,
- pozwala na szybkie i proste tworzenie aplikacji desktopowych,
- wyświetlanie niestandardowych kontrolek jest oparte na GDI+, co skutkuje niską wydajnością,
- technologia nie jest przystosowana do dynamicznego zmiany layoutu,
- aktualnie wypierana przez WPF, który będzie (być może) wypierany przez UWP.

Wybrane technologie

UI – WPF

- windows Presentation Foundation – wprowadzone w .NET Framework 3.0,
- posiada wbudowane mechanizmy do bardziej wyrafinowanych operacji graficznych, transformacji, renderingu 3D, obsługi przeźroczystości,
- o wiele większe wsparcie (w porównaniu do Windows Forms) przy tworzeniu dynamicznych layoutów (bardzo ważne przy internacjonalizacji aplikacji),
- wykorzystuje sprzętowe wsparcie do renderowania zawartości (DirectX). Jednak bez niego wymaga dużych zasobów i może działać wolniej,
- interfejs użytkownika jest definiowanych w XAML-u.

Wybrane technologie

UI - Universal Windows Platform

Wybrane technologie

UI – ASP.NET

- aplikacja uruchamiana jest w środowisku Windows IIS (*Internet Information Services*),
- aplikacja jest dostępna przez dowolną przeglądarkę WWW,
- w porównaniu do aplikacji typu rich-client:
  - użytkownicy nie muszą nic dodatkowo instalować,
  - aplikacja dostępna jest na dowolnej platformie,
  - wszelkie zmiany muszą być aktualizowane tylko po stronie serwera.
- Infrastruktura ASP.NET przewiduje kilka sposobów tworzenia aplikacji webowych

Nowości w .NET 8.0

Performance

.NET MAUI

.NET 8

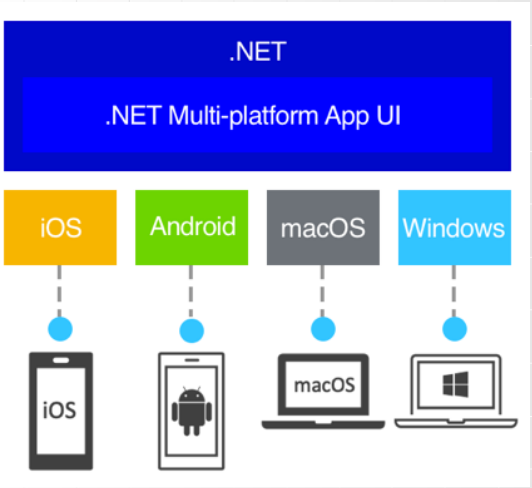
Cloud Native

Artificial Intelligence

ASP.NET Blazor Full Stack

Wybrane technologie

MAUI – Multi-Platform Application User Interface



Interfejs użytkownika aplikacji wieloplatformowej platformy `.NET` (`.NET MAUI`) to międzyplatformowa platforma do tworzenia natywnych aplikacji mobilnych i klasycznych przy użyciu języków C# i `XAML`.

Testy jednostkowe

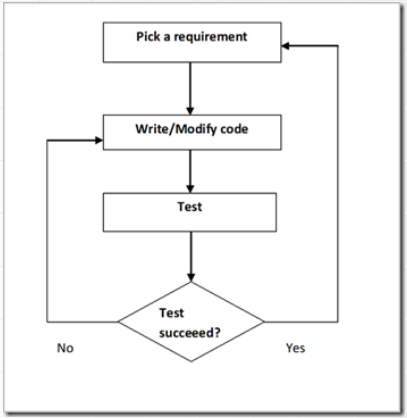
Pisanie testów jednostkowych jest bardziej sztuką projektowania niż weryfikacji. Co więcej, sztuka pisania tego rodzaju testów ma więcej wspólnego z dokumentowanie niż sprawdzanie, oprogramowania. Pisanie testów jednostkowych zamyka wiele pętli sprzężeń zwrotnych, z których zaledwie jedna ma związek z samą weryfikacją.

Rodzaje testów jednostkowych

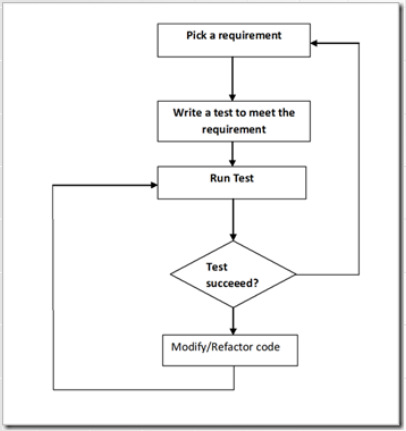
- analiza ścieżek,
- użycie klas równoważności,
- testowanie wartości brzegowych,
- testowanie składniowe.

Wszystkie testy muszą przejść zawsze pozytywnie.

Różne podejścia do projektowania aplikacji



Rysunek 1: tradycyjne podejście



Rysunek 2: TDD

https://www.codeproject.com/Articles/47747/Test-Driven-Development-TDD-Step-by-Step-Part-In

Wytwarzanie sterowane testami

Test Driven Development

- testy automatyczne dla całego systemu
- łatwiejsze tworzenie nowych funkcjonalności (stare pozostają stabilne i przetestowane),
- większa pewność dla deweloperów (można spokojnie refaktoryzować),
- skrócony czas testów manualnych,
- sposób dokumentacji systemu,
- sposób na fixowanie bugów,
- sposób na szybką detekcję powracających bugów w przyszłości.