Update Days
Implement AI

Co se událo ve světě .NETu za posledních 10 let?

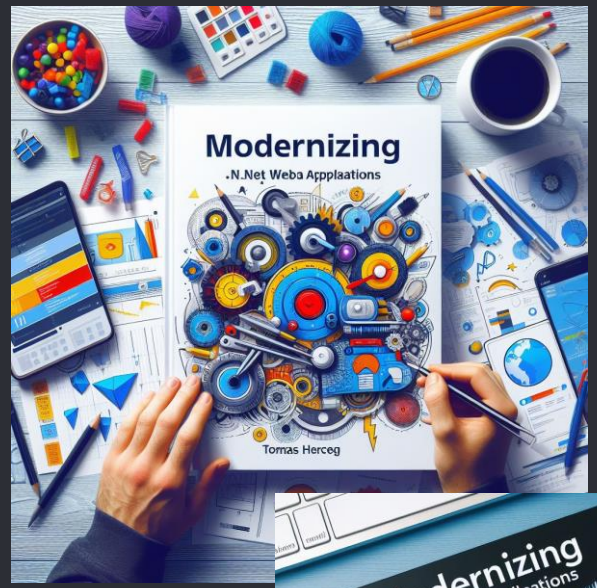
Tomáš Herceg

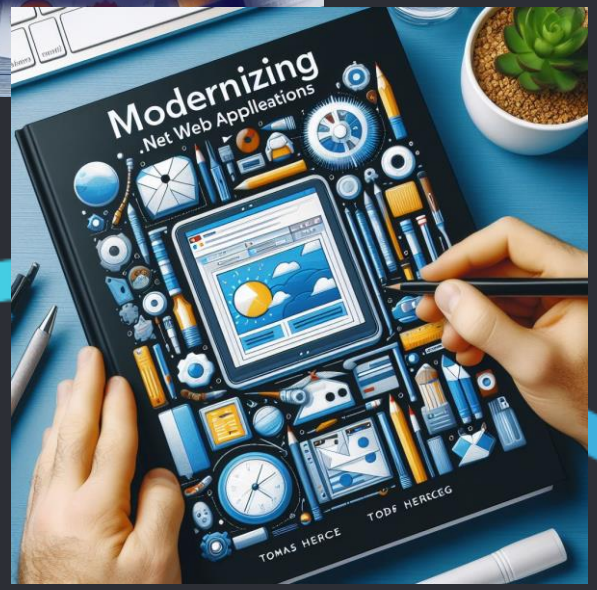
CEO @ RIGANTI

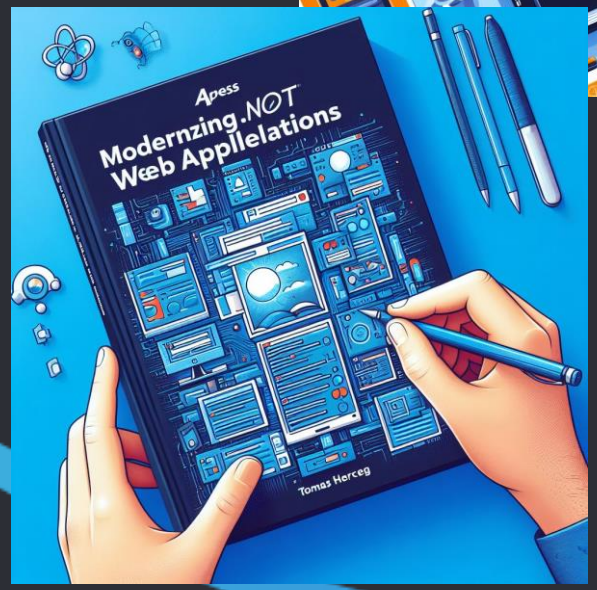
Microsoft MVP

tomasherceg.com

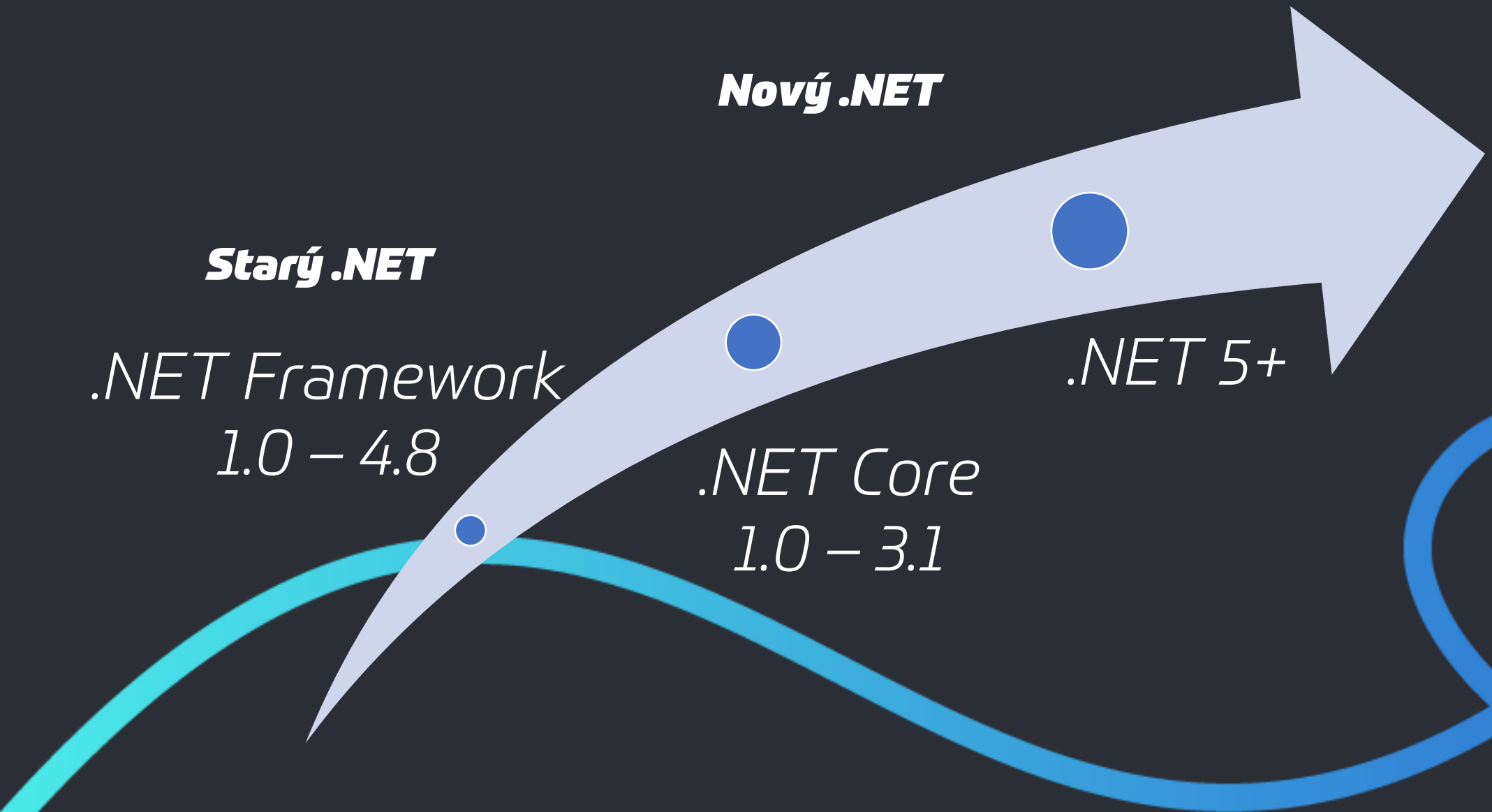
Proč tato přednáška?







Trocha historie



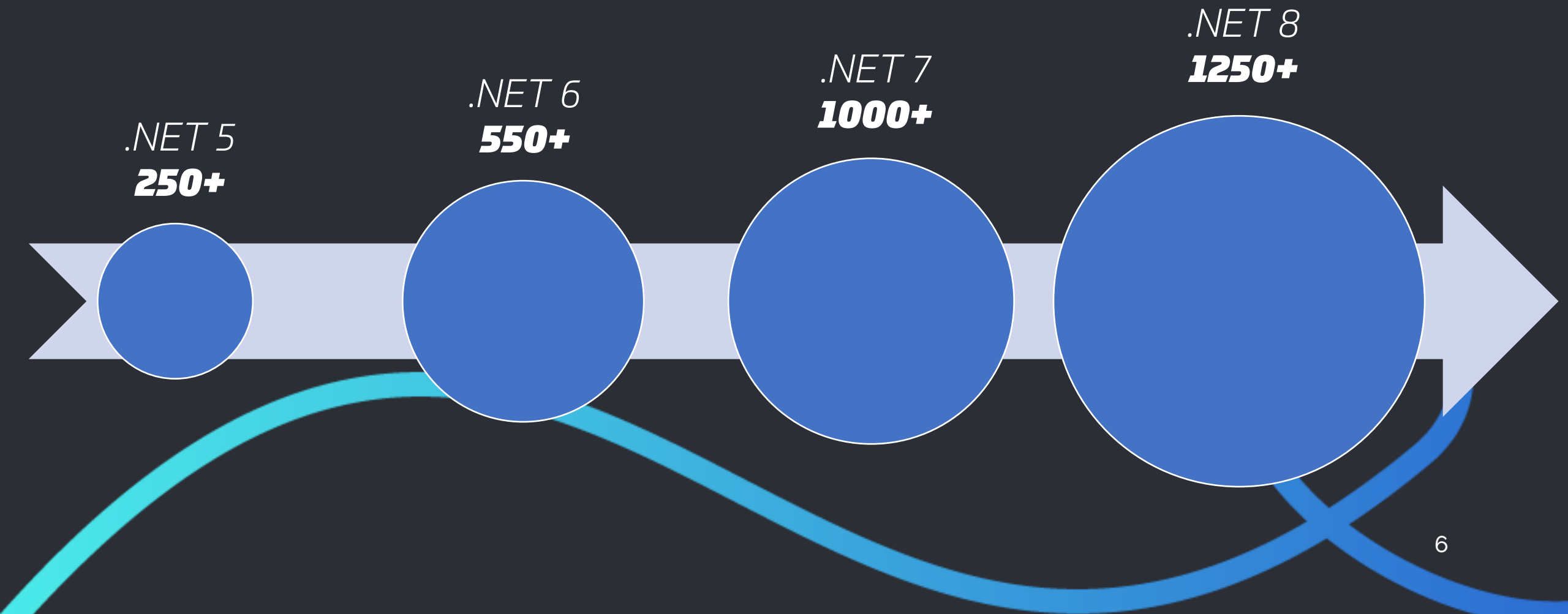
Nový .NET

- *Je s námi již 8 let*
  - *Ano, už takhle dlouho*
- *Obrovské množství změn a novinek*
- *Multiplatformnost, nižší nároky na hardware*

Výhody nového .NETu

- ***Performance***
- *Produktivita při vývoji*
- *Vylepšení architektury*

Počty PR zabývající se performance



Vrstvy, kde se dá optimalizovat

Aplikační kód

BCL

Runtime

Just-In Time Compiler

- *MSIL kompiluje do machine kódu pro daný procesor*
- *Rychlost kompilace × Rychlost vykonání kódu*

- **Tiered Compilation** – *pomalá a rychlá varianta*
- **On-stack Replacement** – *cykly s více iteracemi*
- **Profile Guided Optimization** – *sbírání telemetrie*

Jak počítat iterace

- *OSR a PGO závisí na počtu iterací v cyklu*

- *Není potřeba úplně přesný výsledek – stačí přibližný*
- *Bez locků se ztrácelo hodně iterací*
- `Interlocked.Add` *je zbytečně pomalé*

- *Správné řešení není triviální*

Inlining

- *Rozkopírování těla metody na místa, odkud se volá*
  - *Méně skoků v paměti*
  - *Nemusí se volat metoda*
  - *Hodnoty některých parametrů jsou známé*
- *Nesmí se to ale přehánět*

Inlining

```
var text = FormatNumber(someNumber, "f");
Console.WriteLine(text);

string FormatNumber(int value, string format)
{
    switch (format)
    {
        case "f":
            return DoSomethingWithValue(value);
        case "d":
            return DoSomethingElseWithValue(value);
        default:
            return DoFallback(value);
    }
}
```

Inlining

- *Nový .NET je v tom mnohem lepší*
- *Umí provádět tzv.* **constant folding**

- `new DateTime(2023, 9, 1)`
  ➔ `new DateTime(0x8DBAA7E629B4000)`

Další magie

- *Rušení bound checkingu tam, kde to není nutné*
- *Náhrada % za násobení bitové posuny*
- *Eliminace podmínek a větvení*

Garbage Collector

- *Náhrada velkých segmentů za menší regiony*
  - *Segment nelze uvolnit, pokud je v něm pinned object*
- *Náhrada unmanaged kódu managed kódem*
  - *Nepřímý efekt – GC nemusí čekat na vlákna v umanaged sekci*

Span<T> a Memory<T>

- *Span – odkaz dovnitř pole*
  - *Kdekoliv - na heapu, na stacku, v unmanaged paměti*
  - *Provádí se bound checking*

- *Ref struct*
  - *Nemůže existovat na heapu ani být zaboxován*
  - *Nelze vrátit ven z metody nebo předávat dál*

K čemu to je?

- *Ušetření alokací paměti*
  - `str.Substring` *vytváří novou instanci*
- *Představte si, že píšete parser*
  - *Máte jeden dlouhý* `string` *se vstupem*
  - *Pro zavolání* `double.TryParse` *potřebujete vyrobit* `string`

```
ReadOnlySpan<char> span = inputString.AsSpan();

if (double.TryParse(span.Slice(currentIndex, tokenLength), ...
```

ValueTask<T>

- *Instance třídy* `Task` *zabírá v paměti ~80 bajtů*
  - *Pro určité výsledky má .NET cacheované instance*
- *Async call nemusí být vždy asynchronní*

- `ValueTask` *je struktura*
  - *Pokud operace skončila úspěšně a synchronně, jen hodnota*
  - *Jinak reference na plnohodnotný* `Task`

Source Generators

- *Spousta knihoven generuje při startu kód dynamicky*
  - *Dependency Injection kontejnery*
  - *AutoMapper*
  - *Entity Framework*
  - *JSON a jiné parsery*
- *Přes* `Reflection.Emit` *nebo LINQ Expressions*

Source Generators

- *Kód se generuje v době kompilace*
  - `System.Text.Json`
  - *Volání COM metod a* `DllImport`
  - `Regex`

System.Text.Json

- *Allocation-free parser a serializer*
- *Hojně využívá* `Span`
- *Nativní podpora UTF-8*
  - *Nemusí se konvertovat na* `string` *a zpátky*

Parsování a formátování

- *I dnes se objevují nové algoritmy*
  - *Eisel-Lemire pro rychlejší parsování desetinných čísel*
  - *Euklidovské afinní funkce pro převody data a času*

- *Přepsáno, aby využívalo* **Span**

Vektorizace

- *SIMD – instrukce pro práci s vektory (více hodnotami) najednou*
- *Je třeba zjistit, jestli to CPU podporuje a jak velké vektory umí*
- *Různé metody pro* `Span`
- *Hojně využívá* `string`
  - `IndexOf`, `Substring`, `Replace`
  - *Převod mezi kódováními*

Příklad vektorizace

```
int NaiveIndexOf(byte[] text, byte c)
{
    for (var i = 0; i < text.Length; i++)
    {
        if (text[i] == c)
        {
            return i;
        }
    }
    return -1;
}
```

Příklad vektorizace

```
int NaiveVectorizedIndexOf(byte[] text, byte searchedChar)
{
    var textSpan = text.AsSpan();
    var start = 0;
    if (Vector.IsHardwareAccelerated)
    {
        var searchedVector = new Vector<byte>(searchedChar);
        while (start + Vector<byte>.Count <= textSpan.Length)
        {
            var textVector = new Vector<byte>(textSpan.Slice(start));
            if (Vector.EqualsAny(textVector, searchedVector))
            {
                break;
            }
            start += Vector<byte>.Count;
        }
    }
    ...
}
```

Kolekce

- *Všechny kolekce se optimalizovaly*
  - `List`, `Dictionary`, `Queue`, `Stack`, `HashSet`, …
- *Máme čtyři typy kolekcí*
  - *Standardní*
  - *Concurrent*
  - *Immutable – optimalizované na rychlé kopírování a modifikaci*
  - *Frozen – optimalizované na rychlé čtení*

Více informací

- *Blogposty od Stephena Touba*
  - *.NET 8*
  - *.NET 7*
  - *.NET 6*
  - *.NET 5*
  - *.NET Core 3.0*
  - *.NET Core 2.1*
  - *.NET Core*

Výhody nového .NETu

- *Performance*
- ***Produktivita při vývoji***
- *Vylepšení architektury*

Větší výběr vývojových prostředí

- *Visual Studio*
  - *Stále nejoblíbenější*
  - *Updaty cca každé 3 měsíce – přinášejí zásadní zlepšení*
- *Visual Studio Code*
  - *OmniSharp nebo C# Dev Kit (licencovaný)*
- *JetBrains Rider*
- ~~*Visual Studio for Mac*~~ 

Rychlejší projektový systém

- *Co ukazuje Solution Explorer vs co je ve filesystému*

- *Nový projektový systém je psaný v .NETu*
- *Starý byl psán v C++ a používal COM – pomalé*

Hot Reload

- *Patchování běžícího procesu změnami v kódu*
- *Každá verze .NETu podporuje složitější operace*
  - *Změna kódu uvnitř metody – jednoduché*
  - *Přidávání a úpravy memberů ve třídách*
  - *Úpravy generických typů*
  - *Změna návratového typu metod*
  - *…*

Lepší zacházení s NuGetem

- *Staré projekty měly v solution složku* `packages`
- `packages.config` *obsahoval i tranzitivní reference*
  - *Jak poznáte, které balíčky už nejsou potřeba?*
- *Binding redirecty* 🤢

- `PackageReference` *lze používat i ve starých projektech*

SDK-style projektové soubory

- *Syntaxe* `.csproj` *je mnohem jednodušší*
- *Mnohem méně merge konfliktů v Gitu*
- *Snadné ruční úpravy*
  - *Doporučuju udržovat pořádek a stručnost*

- *Automatický import build targets z NuGet balíčků*

Výhody nového .NETu

- *Performance*
- *Produktivita při vývoji*
- ***Vylepšení architektury***

Konfigurace

- `ConfigurationManager` *je statická třída*
  - *Obtížné používání v testech*

- `ConfigurationBuilder` *a* `IConfiguration`
  - *Možnost napojit na různé zdroje – JSON, environment, User Secrets*
  - *Možnost sledovat změny konfigurace za běhu*

Dependency Injection

- *Pro .NET Framework existovala spousta DI kontejnerů*
  - *Castle Windsor, Unity, Autofac, SimpleInjector…*
- `IServiceCollection` *a* `IServiceProvider`
  - *Možná integrace s pokročilejšími DI kontejnery*
  - *Na registrace by convention se hodí knihovna Scrutor*

Logging

- *Na .NET Frameworku třídy* `Trace` *a* `TraceListener`
  - *Opět statické – špatně se s tím pracuje v testech*
  - *Autoři knihoven měli těžký život*

- `LoggerFactory` *a* `ILogger<T>`
  - *Strukturované logování (kolekce klíč-hodnota)*
  - *Snadná integrace s externími službami*
- `Activity` *a* `Metric` *– podpora metrik a counterů*

Hosting

- *Jak spustit .NET Frameworkovou webovou aplikaci bez IIS?*

- *Jednotný kód v* `Main` *pro různé typy aplikací*
  - *Zajišťuje konfiguraci, logování, DI*
  - *Error handling, graceful termination…*

Další extensions

- *Balíčky* `Microsoft.Extensions.*`

- `Caching`
- `Resilience`

Jak dostat svou aplikaci na nový .NET?

- *Není to vždy jednoduché – kniha bude na podzim*
- *Desktop, knihovny*
  - *.NET Upgrade Assistant*
- *Webové aplikace*
  - *Side-by-side migrace pomocí YARP*
  - *In-place migrace pomocí DotVVM*

Update Days
Implement AI

Implement AI into software

4 - 5 April 2024
Prague / Online

Update Days
Implement AI

Q&A

hercegtomas tomasherceg tomasherceg