[1- LinkedIn article about Delegates in implementing functional paradigm](https://www.linkedin.com/posts/ahmed-b-ramzy_delegates-in-c-a-gateway-to-the-functional-activity-7287859267942936576-I6uz?utm_source=share&utm_medium=member_desktop)

**Parallel Programming and Concurrency**

In the world of C#, **parallel programming** and **concurrency** are key concepts that help you execute multiple operations simultaneously, improving the efficiency and performance of your applications.

* **Concurrency** is about managing multiple tasks at once. Think of it as a waiter handling orders from multiple tables; they're not doing everything at the same time but are efficiently managing multiple tasks.
* **Parallelism** takes it a step further by actually performing multiple operations at the same time. Imagine multiple chefs cooking different dishes simultaneously in a kitchen.

In C#, you can implement these concepts using the System.Threading and System.Threading.Tasks namespaces, which provide classes like Task and Parallel to help you write concurrent and parallel code.

**2- Asynchronous Programming with async and await**

Modern applications often need to perform tasks without freezing the user interface or blocking other operations. This is where **asynchronous programming** comes in, and in C#, it's made simple with the async and await keywords.

* **async** is used to declare a method as asynchronous.
* **await** is used to pause the execution of an async method until the awaited task completes, without blocking the main thread.

By using async and await, you can write code that remains responsive, especially when dealing with I/O-bound or long-running operations.

Sources

1. <https://www.c-sharpcorner.com/article/difference-between-concurrency-and-parallelism-in-c-sharp/>

2- <https://learn.microsoft.com/en-us/dotnet/csharp/asynchronous-programming/>

3- <https://learn.microsoft.com/en-us/dotnet/csharp/asynchronous-programming/async-scenarios>

3 - **Unit Testing and Test-Driven Development (TDD)**

Ensuring your code works as intended is crucial, and that's where **unit testing** and **Test-Driven Development (TDD)** come into play.

* **Unit Testing** involves writing tests for small units of your code, like individual methods or classes, to verify they function correctly.
* **TDD** is a development approach where you write the tests *before* writing the actual code. This might sound backward, but it helps in defining clear requirements and leads to more robust and maintainable code.

In C#, frameworks like MSTest, NUnit, and xUnit.net are popular choices for writing unit tests.

Sources

1. <https://blog.coscreen.co/blog/tdd-in-c-guide/>
2. <https://www.codeproject.com/Articles/5374061/Test-Driven-Development-in-Csharp>
3. <https://www.youtube.com/watch?v=dp1WRyR0TiE>