Monorepo vs. Monolith: A Comparative Study in .NET Applications

Nagib Sabbag Filho

Leaders.Tec.Br, 2(25), ISSN: 2966-263X, 2025.
e-mail: profnagib.filho@fiap.com.br
DOI: https://doi.org/10.5281/zenodo.15787068
PermaLink: https://leaders.tec.br/article/828c25
Received: 19 Jun 2025 / Accepted: 21 Jun 2025 / Published online: 23 Jun 2025

Abstract:

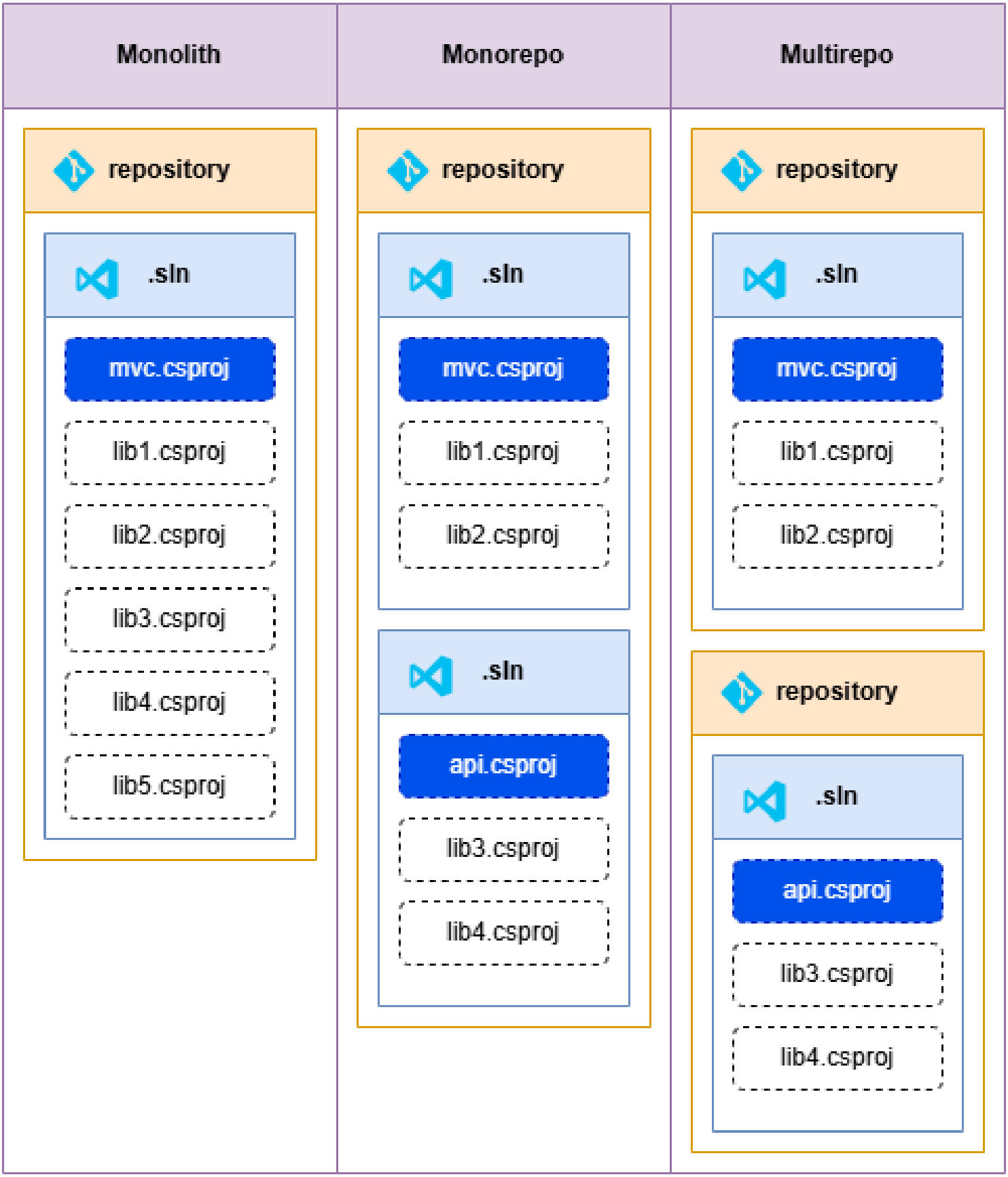
This article explores the monorepo and monolith software development approaches, focusing on their characteristics, advantages, and disadvantages. The monorepo centralizes multiple projects in a single repository, facilitating dependency management and collaboration, but it can lead to scalability and complexity challenges.

Key words:

Monorepo,Monolith,.NET Applications,Software Development,Software Architecture,Code Management,Versioning,Scalability,Maintenance,Continuous Integration,Continuous Delivery,Microservices,Modularity,Collaboration,Build Tools,Performance,Automated Testing,Dependencies,Performance,Project Structure.

Introduction to Concepts

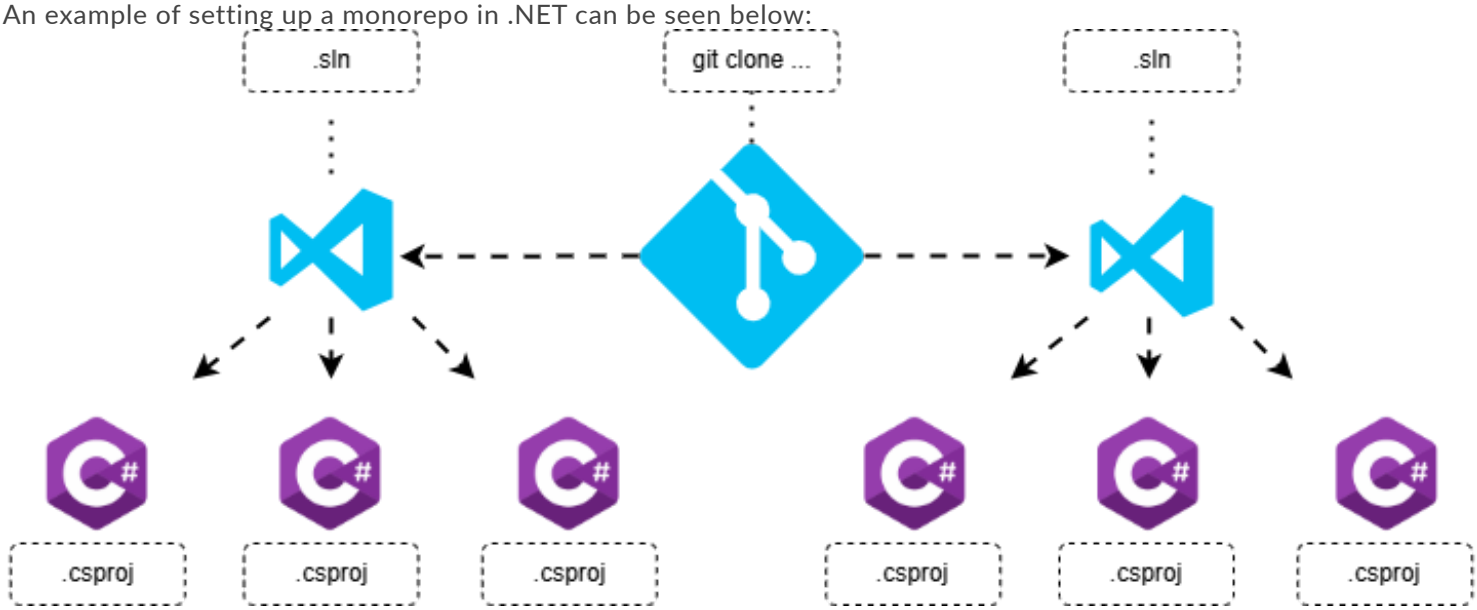
In recent years, software development has rapidly evolved, bringing new approaches and architectures for building applications. Among these approaches, monorepo and monolith stand out as popular options, especially in .NET environments. This article addresses the characteristics, advantages, and disadvantages of each of these strategies, allowing for a deeper understanding for software developers and architects.



Introduction to Monorepo

A monorepo, or monolithic repository, is an approach in which multiple projects and packages are stored in a single repository. This strategy is commonly used by large organizations seeking to simplify dependency management and promote collaboration among teams (TONIN, 2024). By adopting a monorepo, companies can take advantage of a structure that allows interaction and synergy among different projects, reducing duplication of efforts and increasing efficiency (SHAKIKHANLI; BILICKI, 2024).

An example of setting up a monorepo in .NET can be seen below:



Advantages of Monorepo

One of the main advantages of a monorepo is the ease of dependency management. With all projects in one place, it is simpler to ensure that they are all using compatible versions of common libraries (TONIN, 2024). Moreover, collaboration among teams is facilitated, as changes in one project can be easily integrated into others. This is particularly useful in large teams working on multiple components of a complex application.

Another positive point is the consistency in coding style and tools used, as all developers are working within the same repository. This promotes a more cohesive and integrated coding culture, where standards and best practices can be uniformly applied. Thus, code maintenance becomes simpler and more efficient.

Additionally, monorepos allow the use of centralized build and testing automation tools that can be applied across all projects, ensuring that they are always functioning correctly, which increases the reliability of the system as a whole.

Disadvantages of Monorepo

Despite the advantages, using a monorepo comes with challenges. One of the most significant is the increased build and test time, which can become a bottleneck as the repository grows. Tests that were once quick can become slow and costly, especially if the code structure is not optimized. Additionally, management complexity can increase, especially if different teams are working on unrelated parts of the code.

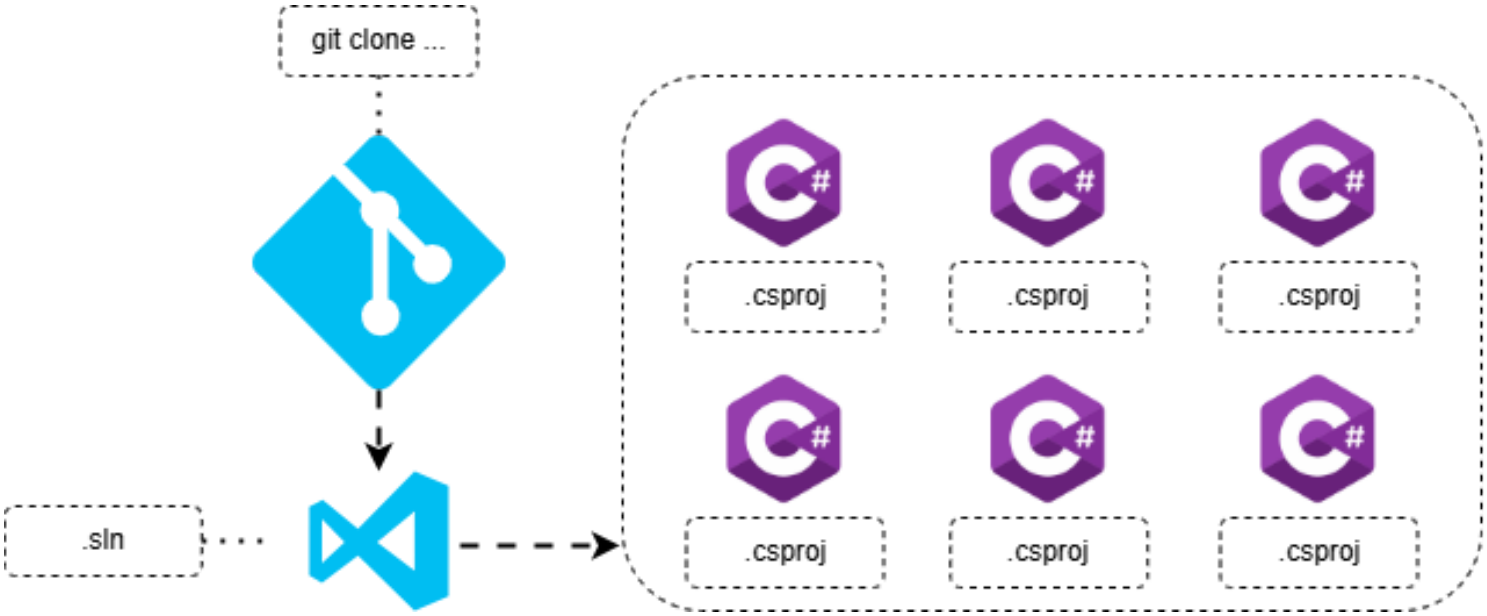
Scalability is another critical point: very large repositories can become difficult to navigate and manage, requiring additional tools and processes to maintain efficiency. In many cases, teams need specific strategies to deal with these difficulties, such as segmenting projects within the monorepo, which can add more layers of complexity.

Lastly, permission and access management can become more complex, as a single repository houses multiple projects that may have different levels of sensitivity and security.

Introduction to Monolith

In contrast, a monolith is an architecture where all components of an application are integrated into a single solution (SABBAG FILHO, 2025). This approach has traditionally been used in .NET applications, where the code is often organized in a single codebase. The monolith is often the initial choice for many startups and small projects due to its simplicity and speed of development.

An example of a monolithic application in .NET can be seen below:



Advantages of Monolith

Monoliths have some clear advantages, especially in smaller projects or early development phases. The simplicity in setup and deployment is a major attraction, as the entire application is deployed as a single unit, reducing the complexity of the production environment. This also reduces the time needed to set up and manage development and testing environments.

Furthermore, monolithic applications can perform better in some situations, as all parts of the system are within the same process, allowing for faster function calls between components. This is particularly important in applications where latency and speed are crucial.

Another advantage is that the monolithic architecture makes monitoring and log analysis easier, as all application interactions are centralized in one location. This facilitates problem identification and debugging when failures occur.

Disadvantages of Monolith

However, monoliths also present significant disadvantages. Scalability can be a problem, as the entire application must be scaled, even if only a part of it is under load. This can lead to inefficient resource usage, where, for example, a component that needs to be scaled to handle an increased load may force the entire application to be replicated, resulting in unnecessary costs.

Moreover, maintenance can become difficult as the application grows, as small changes can have impacts on various parts of the system. This interdependence between components can result in increased development and testing time, as any change may require extensive validation.

The dependency on a single technology can limit the team's ability to adopt new technologies, as changing one part of the monolith may require modifications across the entire codebase, making technological evolution a slower and more complicated process.

Direct Comparison: Monorepo vs. Monolith

When comparing monorepo and monolith, it is essential to consider the context in which each is used. The monorepo is more suitable for projects involving multiple interdependent components or services, while the monolith may be the ideal choice for smaller applications that do not require such complexity. The choice between a monorepo and a monolith should be guided by the specific needs of the project.

Additionally, while the monorepo facilitates collaboration among teams, the monolith may offer a lower learning curve for new developers, as all the application logic is consolidated in one place (JASPAN et al., 2018). This can be particularly advantageous in teams that are forming or in environments with high personnel turnover.

Another point to consider is continuous integration: a monorepo can allow for smoother integration between different parts of the code, while a monolith may require more rigorous testing to ensure that changes in one part do not break functionalities in another.

Use Cases and Scenarios

In a scenario where a team is developing a large-scale application with multiple services (e.g., an order management system and an inventory system), a monorepo may be the ideal choice. This approach allows teams to work on different services in a coordinated manner, applying changes across the application more efficiently.

On the other hand, if the team is building a simple task management application, a monolith may be more practical. In smaller projects, the additional complexity of a monorepo may not justify the benefits, making the monolith a more logical choice.

Furthermore, many organizations are adopting hybrid approaches, where parts of an application may be developed as a monolith while others are extracted into a monorepo. This allows for the best of both worlds, where complexity and scalability can be managed more effectively.

An example of this is an application that has a central monolithic module responsible for essential functionalities, while auxiliary modules that are more dynamic or change frequently are developed as independent services in a monorepo. This strategy offers flexibility and allows teams to adopt new technologies as needed.

Final Considerations

Both approaches, monorepo and monolith, offer advantages and disadvantages that must be carefully considered before making a choice. Understanding the project's needs, team structure, and desired scalability are crucial factors in determining which approach will be more effective. As technology continues to evolve and development practices transform, the choice between monorepo and monolith may very well depend on the specific context of each project.

Additionally, it is important for teams to be open to revisiting their choices over time. As a project grows and its needs change, what may have been an ideal choice at an early stage may no longer be the best option. Therefore, continuously monitoring and evaluating the architecture and code structure is essential to ensure that the project remains efficient and sustainable in the long term.

Finally, adopting an agile and iterative development approach can help mitigate many of the issues associated with both monorepo and monolith. By allowing teams to experiment, learn from failures, and adjust their development practices, organizations can make the most of the opportunities each approach offers.

References

• TONIN, Rangel Cristiano. Centralized source code strategies for efficient management of software product variants. 2024.

• SABBAG FILHO, Nagib. Comparative Analysis between Monolithic Architecture and Microservices in .NET Applications. Leaders Tec, v. 2, n. 13, 2025.

• SHAKIKHANLI, Ulvi; BILICKI, Vilmos. Optimizing branching strategies in Mono-and Multi-repository environments: A comprehensive analysis. Computer Assisted Methods in Engineering and Science, vol. 31, no. 1, pp. 81-111, 2024.

• JASPAN, Ciera et al. Advantages and disadvantages of a monolithic repository: a case study at Google. In: Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice. 2018. p. 225-234.

Nagib is a University Professor and Lead Systems Architect, with a career marked by solid academic and professional achievements. He holds relevant technical and agile certifications, such as GitHub Copilot, the PSM1, and Azure Fundamentals. He has a postgraduate degree from SENAC and the Mackenzie Presbyterian University, with an MBA in Software Technology from USP, in addition to participating in extension programs at MIT and the University of Chicago. He is the author of a peer-reviewed scientific article on chatbots, presented in person at the University of Barcelona. He is actively involved in the technical community, with over 50 published articles, including more than 10 articles on iMasters. He has also been a speaker in the .NET Architecture track at TDC São Paulo 2024 and in the Solutions Architecture track at TDC Floripa 2025.