I’m familiar with both the agile development method and the waterfall/traditional development method, but I’ve never thought about their impact on software architecture. The traditional and agile development methods are very different, it’s not surprising to me that these development methods typically rear architecturally different products.

Agile vs Traditional Development

The traditional development method consists of 5 steps completed in order: Requirements analysis, Design, Implementation, Coding and Testing, Maintenance. All requirements are to be gathered before design is developed, the design should be completed before implementation, implementation before coding and testing and so on.

The agile development method takes a more iterative approach. It consists of these six steps performed in a loop: Planning, Requirement Analysis, Designing, Implementation, Testing, Deployment. Agile tends to be combined with other development methods like feature driven development, kanban, scrum, lean development, etc. But what they all have in common is the iterative approach to software development. (GeeksForGeeks, 2023)

Coupling and Modularity

Because the traditional development style is linear and sequential, it may result in a tightly coupled product. This may work out for the system since it was developed as a whole unit. An example of such an architecture pattern would be a monolith. Because all the requirements are gathered prior to development, additional modularity may be seen as unnecessary and over-engineered. Having additional level of analysis and design prior to development can help reduce over-engineering for functionality that may never be needed.

Due to the iterative development style of agile, this development method supports modularized and component-oriented architecture well. Breaking the application into modules of like functionality allows for asynchronous development by multiple teams, loose coupling of functionality within the larger application, and reutilization of previously developed code. There are many examples of this architecture pattern, the most popular being microservices.

Documentation

The traditional development style strongly supports a well-documented final product. Because of the amount of analysis and design required prior to development, the product should be well documented before it's even built. Only minor changes throughout the development process should need additional recording, potentially speeding up the development process.

Agile development, on the other hand, tends to create a lot of documentation that needs constant updating. Because requirements change throughout the development process and functionality is released in bite-sized features, code is constantly changing. For agile, it's a good idea to have codebase driven documentation like swagger for API documentation, and strong emphasis on comments in code with linting rules for code style.

Extensibility & Scaling

This goes hand in hand with coupling and modularity. Because traditional style development usually results in a system that was built in a tightly coupled architectural structure, it is usually not open to extensibility by default. Scalability and extensibility of a system built using the traditional style is highly dependent on the original design of the product.

A system created using Agile methodology focuses on scalability and extensibility as a cornerstone of their development structure. Because requirements change regularly, the architecture must remain adaptive throughout the entirety of the software development life cycle. This ability to react to changes quickly and efficiently is what gives "Agile" its name. (Agile Alliance, 2001)

Automation & CI/CD

Because the traditional development style is focused on releasing large features at a time, or potentially the entire product, there isn't much of a need for CI/CD pipelines. Additionally, large scale automated testing isn't as much of a focus since there isn't a need for as much regression testing as with the agile method. Also, linear and sequence-based development minimizes the code changes after initial development, reducing the need for regression testing. This development pattern tends to support manual testing over automation because of this. Because of the reduced need for automated testing and CI/CD pipelines, this is a significant reduction in the amount of development work required.

Agile development pretty much requires automated testing and CI/CD pipeline development. Because of the iterative development style, code is constantly changing. These updates can result in breakages of pre-existing functionality, hence the need for automated regression testing. Also, mini features are deployed constantly in agile development, which is best handled using a CI/CD pipeline. (GeeksforGeeks, 2024)

Conclusion

Ultimately, I think the development method chosen comes down to what is desired of the product. Is it supposed to be a long maintained, adaptable, large piece of software? Then maybe the modularity, scalability, and extensibility of the Agile development practice is the right choice. Is it a product that, once it's built, is not expected to ever change? Then maybe it's best to use a waterfall approach to get it right the first time without spending unneeded cycles.

Agile Alliance. (2001). *Manifesto for Agile Software Development*. Agile Manifesto. https://agilemanifesto.org/

Difference between Agile Testing and Waterfall Testing. (2024, July 12). GeeksforGeeks. https://www.geeksforgeeks.org/difference-between-agile-testing-and-waterfall-testing/

GeeksForGeeks. (2023, December 5). Difference between Traditional and Agile Software Development. GeeksforGeeks. https://www.geeksforgeeks.org/difference-between-traditional-and-agile-software-development/