"Call by value" is a strategy used by certain programming languages for passing parameters and evaluating expressions; this strategy affects the flow of data, runtime complexity, and program design.

The call by value method ensures functions operate on data copies, not original values, which preserves data integrity from the calling environment. Parameters are evaluated once, and their values are copied into the function's runtime environment, thereby isolating the function's scope and execution from external variables. This copying, which stores values typically in the stack or registers, prevents any manipulation within the function from affecting the original data. From a theoretical standpoint, this isolation is key to enhancing code predictability and safety, facilitating easier debugging and testing due to the encapsulation of function executions. Call by value offers significant advantages in terms of safety and data localization, reducing risks of unintended side effects and data corruption. This method is particularly valuable in environments that demand high reliability and data integrity, such as embedded systems where preserving the original data state is crucial.

The call by value and call by name parameter passing strategies, tailored for different programming scenarios, exhibit distinct strengths and weaknesses. Call by value ensures consistent argument values throughout a function's execution, enhancing reliability and predictability. Conversely, call by name delays the evaluation of arguments until they are actually used within the function, optimizing resource usage and potentially improving performance in scenarios where not all arguments are used. However, this approach can compromise reliability because re-evaluating arguments may yield different results if the underlying data changes. In terms of developer experience, call by value offers simplicity and is easier to debug, making it a preferred choice for many developers, while call by name's lazy evaluation can lead to complexities in debugging, particularly in managing state changes. Scalability wise, call by value may struggle with large data loads due to its copying mechanism, whereas call by name can scale more effectively by avoiding unnecessary evaluations. Ultimately, the choice between call by value and call by name hinges on the specific requirements of the application, balancing efficiency and simplicity against performance needs and the complexity of managing state changes.

The parameter passing mechanism of call by value, foundational in many programming languages, notably ALGOL in the 1950s, has significantly evolved since its inception. Initially appreciated for simplifying function behavior reasoning, this mechanism became a staple in procedural programming languages such as C, influencing the development of object-oriented and functional languages by promoting safer and more predictable code. It is crucial in systems requiring high reliability and maintainability. As modern computing has demanded more robust concurrency and scalability, the relevance of call by value has been reaffirmed, particularly in multithreaded and distributed environments. The design axis has shifted towards optimizing this mechanism to meet the challenges of modern software demands, focusing on reducing memory overhead and enhancing performance, particularly crucial for adapting call by value to high-performance and cloud computing needs. These developments include techniques like copy-on-write, which minimizes the performance costs while maintaining the semantic benefits of call by value.

1. Maraist, John, et al. "Call-by-Name, Call-by-Value, Call-by-Need, and the Linear Lambda Calculus." *Electronic Notes in Theoretical Computer Science*, Elsevier, 25 Sept. 2004, www.sciencedirect.com/science/article/pii/S1571066104000222?ref=pdf_download&fr=RR-2&rr=88112b98ec86224c.
2. Valarcher, Pierre. *Call-by-Name versus Call-by-Value in Primitive Recursion: Storage ...*, www.dicosmo.org/WRS05/proceedings/pierre.pdf. Accessed 2 May 2024.
3. Wadler, Philip. *Call-by-Value Is Dual to Call-by-Name*, homepages.inf.ed.ac.uk/wadler/papers/dual/dual.pdf. Accessed 2 May 2024.