Relationshipility of -ilities

Dirk Engel, info@engel-internet.de, February 29, 2024
https://github.com/dirkengel/

Introduction

If you are involved in software architecture, it is crucial to consider software quality attributes, often referred to as the *-ilities*. Caring for these attributes requires managing the trade-offs among factors that frequently influence each other. This must be done within the specific context for which the software architecture is designed. However, understanding the general relationships between commonly relevant quality attributes can be beneficial.
I came across two sources that provide informative yet broad discussions about relationships among these *-ilities*: *Software Requirements* (Wiegers13) and *Software Architecture in Practice* (Bass21). Figure 1 displays the origin/destination matrix from Wiegers13, sorted according to the ISO 25010 model (ISO), and incorporates additional relationship data from Bass21, particularly those related to *Energy Efficiency*. My three takeaways from interpreting the resulting matrix are: *The Confirmation*, *The Surprise*, and *The Challenge*.

The Confirmation

Performance (Efficiency) comes with negative impacts on many other quality attributes, reinforcing the maxim: Optimize performance only if and where necessary; investigating *Functional Appropriateness* should provide clarity on this matter. The addition of the *Energy Appropriateness* aspect by Bass et al. (Bass21) further sharpens this perspective.

The Surprise

The matrix suggests that *Scalability* comes without any negative side effects; quite the contrast – it appears to be beneficial for several quality attributes. Does this mean every software project should increase scalability? No. In the case of embedded systems, for instance, scaling out is often not possible, and the software has to be prepared to scale up. It is not as simple as running a second instance of a container or process; changes must occur internally. For example, algorithms must support parallelism, and thread pools must be introduced to observe real scaling effects. However, all of this comes at a cost. Once again, a deep look at *Functional Appropriateness* can provide guidance. For me, *Functional Appropriateness* also encompasses aspects like cost appropriateness, complexity, and simplicity. At this point, it is crucial to emphasize that complexity is not the opposite of simplicity. A system might be oversimplified for a given problem, and merely adding complexity will certainly not help.

The Challenge

Verifiability shows the most asymmetric influence on the other attributes; that is, increasing *Verifiability* also helps some other attributes, but increasing these other attributes has a negative influence on *Verifiability*. This kind of relationship is challenging in itself, but I would like to point out another challenge here. Especially when using AI to generate source code, it is desirable to verify that code before human developers start delving into it to fully understand how it works. Furthermore, more source code can be created in less time with the help of AI companions. The generated fragments might be reused and combined in different contexts, and in each new context, the entire setup has to be reverified. Of course, *Verifiability* (and *Analyzability*) were important already in the past, but I believe they become even more relevant for programs not conceived by traditional developer teams but hybrid teams.

References

Bass21	Len Bass et al. “Software Architecture in Practice”, Addison-Wesley, 4th ed., 2021.
ISO	ISO/IEC 25010, https://iso25000.com/index.php/en/iso-25000-standards/iso-25010
Wiegers13	Karl Wiegers, Joy Beatty, “Software Requirements”, Microsoft Press; 3rd ed., 2013.

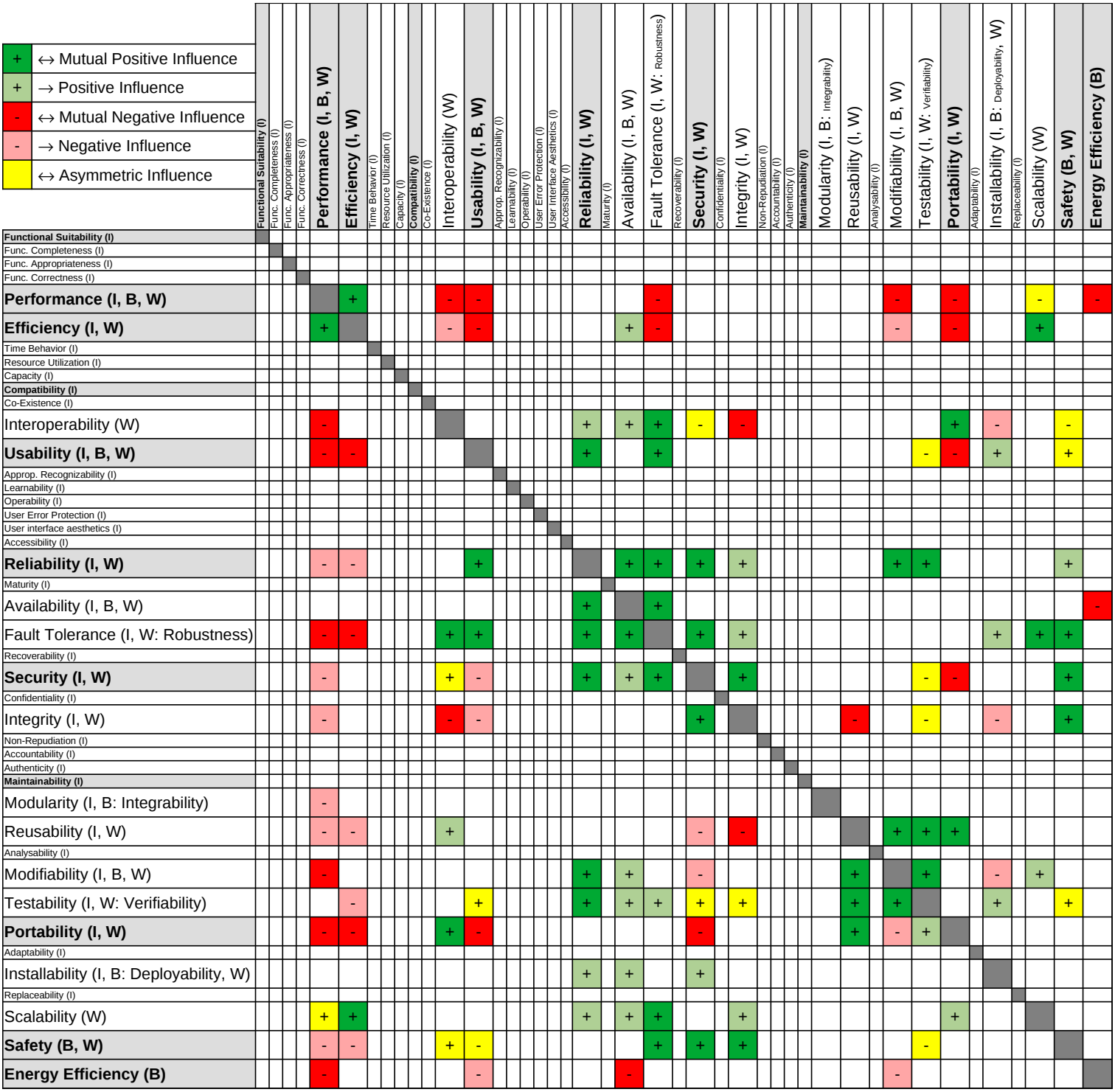

Figure 1: Origin/destination matrix of the SW quality attributes according to ISO 25010 with data from Wiegers13 and Bass21. The capital letters in brackets stand for the corresponding sources: (I)SO, (W)iegers, (B)ass.