

**Review of**  
**Chapman R., Dross C., Matthews S., and Moy Y.**  
**Co-developing Programs and Their Proof of Correctness.**  
***Comm. ACM* 67, 3 (2024) 84–94; 10.1145/3624728.**

T.H. Tse,  
School of Computing and Data Science,  
The University of Hong Kong,  
Pokfulam, Hong Kong

Quality assurance of safety critical software in large industrial applications is challenging due to the complexity of interacting dynamic components and objects. To tackle the issue, the current authors proposed SPARK, which was an “auto-active” method that compromised between fully automatic verification and completely interactive proof assistance. They meticulously designed a programming language, coupled with a “ghost” language for specification and verification. The present article revisits the three decades of research, development, and deployment of SPARK that handle various important aspects such as soundness, completeness, formality, scalability, modularity, and expressiveness.

SPARK supports contracts in ADA2010-like programs, such as pre- and post-conditions, type invariants, and subtype predicates. In response to the notorious pointers in C-like programs, SPARK supports numerous features of pointers in Rust programming, including ownership, borrowing, dynamically allocated heap memory, and lifetime management that ensures references do not outlive the relevant data — thus resulting in memory- and type-safety.

The development team has applied SPARK to various industrial projects, initially at the Bronze level for correct information flows, gradually migrating to the Silver level for the absence of runtime failures, the Gold level for the proof of functional properties, and ultimately aspiring to the Platinum level for full functional correctness.

The article is very well written. It provides straightforward fundamentals to novice designers and implementers, and practical details to experienced co-developers. I highly recommend it to both categories of potential readers.

Various other projects have also addressed the profound issues in dynamically interacting systems using metamorphic testing (MT) [1], which reveals hidden failures by checking the results of multiple executions of the same software. On the one hand, SPARK and MT focus on separate aspects of quality

assurance, namely correctness verification and software testing, respectively. On the other hand, we should not simply regard them as distinct methodologies. In fact, both of them deal with situations where the oracle — or the relation between the expected and actual outputs — are very difficult to verify, as in complex systems with adaptively interacting components and objects. For example, the present authors admit that “proving existentially quantified formulas is hard for provers, as it requires guessing an appropriate value.” To enhance the proposal by the current article for a midway approach between automatic verification and interactive proof assistance, I recommend that the authors consider augmenting the SPARK process with MT, so that the correctness of the target systems can be further improved.

## Reference

- [1] Chen, T.Y.; Kuo, F.-C.; Liu, H.; Poon, P.L.; Towey, D.; Tse, T.H.; Zhou, Z.Q. (January 2018). Metamorphic testing: A review of challenges and opportunities. *ACM Computing Surveys (CSUR)*, 51, 1, 1–27.  
<https://dl.acm.org/doi/10.1145/3143561>