



The economic rationale behind open source

Moving to series production faster through collaborative development

What would happen if suppliers no longer developed their own base software, but instead the industry built on shared code? Eclipse S-CORE aims to reduce dependencies and enable the sharing of development work, thereby changing the rules of the game for automotive software.

Lars Bauhofer

More than 120 active developers from over 20 companies are currently contributing to Eclipse S-CORE, adding up to around 1.9 million lines of code (**Figure 1**). Eclipse S-CORE is today one of the most active projects at the Eclipse Foundation. This illustrates a mechanism that is well known in the software world and which is now also taking hold in automotive development: the “ecosystem flywheel”. The more participants work on a shared codebase, the better the technology becomes. Better technology attracts further programs, whose developments in

turn flow back into the shared foundation.

The key factor here is the scope of each individual development. It is irrelevant who has invested in it: it is subsequently available to all participants. This is why each subsequent rotation of the flywheel becomes easier. For an on-going production program, this is initially just an observation. The relevant question is: what does this actually change in terms of development and safety verification, and what are the consequences for dependencies and cost?

The lock-in outlives standardization

The starting point is a problem that anyone who has ever switched a middleware stack is familiar with. A standard like Adaptive AUTOSAR standardizes the functional interface. However, the behavior underneath it remains vendor-specific, including scheduling, latency profiles, memory footprint, timing jitter, and fault behavior at the limits. A system is designed, measured, verified, and certified precisely against this behavior. Therefore, changing vendors is not simply a matter of swapping com-

ponents. In terms of project effort, it is essentially like starting from scratch.

The lock-in lies one level deeper than the standard (Figure 2), and it can only be resolved where it originates: in the implementation. When all parties execute the same code, the non-functional behavior becomes the industry's shared technical foundation. That is the logic behind Eclipse S-CORE.

Every software product has two types of cost (Figure 3). Development costs are incurred once and include requirements, architecture, implementation, testing, and safety evidence. The development effort for a defined state is limited. Operating costs, however, are ongoing and include certifications, audits, support, defect resolution throughout the product's entire lifecycle.

In the proprietary model, both cost types can be combined. The vendor owns the code and refinances the development costs through licenses and sales volumes. However, in the open-source model described here, this refinancing method is not applicable as published code cannot be invoiced retrospectively. Therefore, development costs must be covered upfront, before the result is published.

That does not apply to operating costs. A community does not guarantee a production version, operate a hardware laboratory or accept liability. This

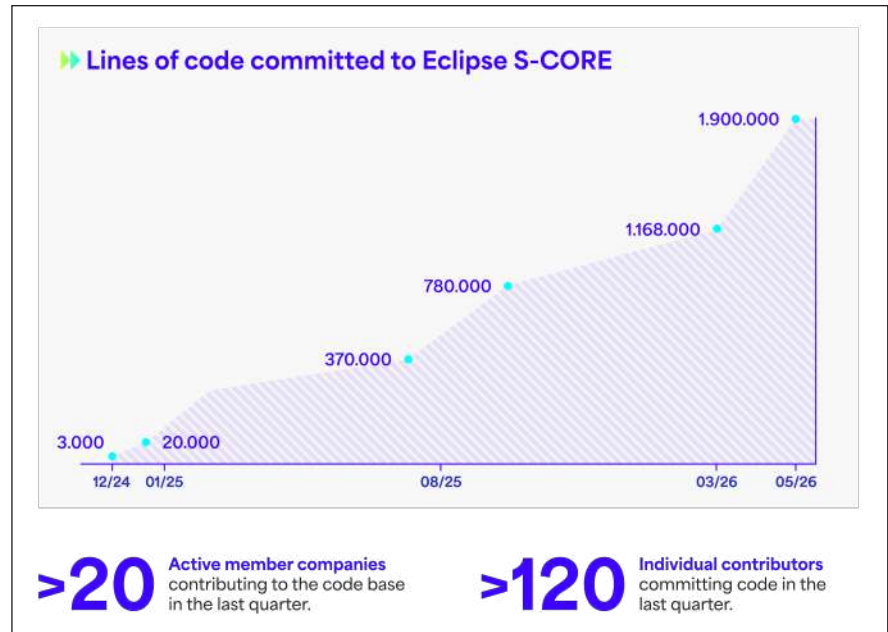


Figure 1: Growth of the Eclipse S-CORE codebase since 2024: over 120 contributors, more than 20 companies, around 1.9 million lines of code. © Eclipse Foundation

remains a paid service in both the open-source and proprietary models. The commercial logic is clear: development is financed in advance, while operation is paid for on an ongoing basis.

How the shared foundation is changing things for developers

The shared implementation and separation of development and operations have four effects that directly impact development projects.

- **Safety verification results become transferable.** In addition to shared code, S-CORE also aims to share certification artifacts. Requirements, architecture, test cases, and safety evidence are produced through an open process, with the aim of achieving a "Safety Element out of Context" certification that all participants can reuse. This enables a substantial part of the safety verification effort to be shared.
- **The supplier can be changed while the platform remains the same.** Switching commercial partners leaves the stack unaffected. Applications and safety verification therefore do not need to be migrated to a new implementation. At the same time, the procurement team can negotiate performance and price without the threat of migration.
- **Every further enhancement works in both ways.** If another participant further develops a component or ports it to a new generation of hardware, the result is available for their own program. Conversely, every in-house development strengthens the foundation on which all others are built, making the platform more attractive to further participants. No one bears the full burden alone in any cycle, and the rate of improvement increases with every new participant. This is exactly where the flywheel effect comes into play, accelerating with every turn.

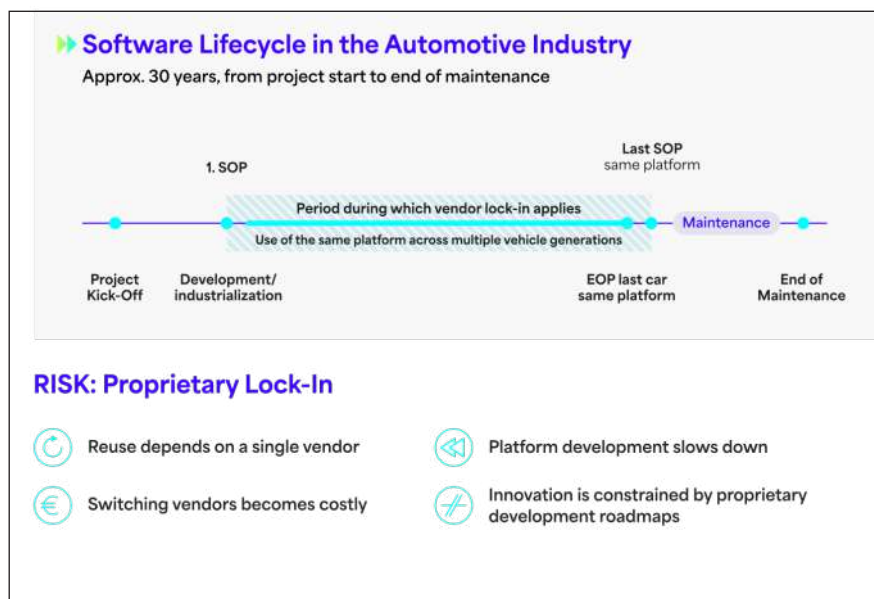


Figure 2: Proprietary lock-in affects the entire 30-year software lifecycle: code reuse becomes dependent on a single vendor, switching becomes expensive, and platform development slows down. © Qorix GmbH



Figure 3: Development costs are one-time expenses, while operating costs are incurred over the entire 20- to 30-year product lifecycle. In automotive software, this difference becomes a decisive cost factor. © Qorix GmbH

- **Developed functionality survives a vendor switch.** Whatever a program funds to close its own gaps in the open-source base remains part of the shared platform. It does not disappear along with the contract.

Taken together, this leads to a simple conclusion. In the proprietary model, every vendor and every OEM must redevelop and finance the foundation from scratch to reach a comparable level of functionality, so the cost per participant remains more or less constant (**Figure 4**). In the open-source model, however, the foundation is built once at the ecosystem level, so the cost per participant can fall while the number of participants, and with it the rate of improvement, increases. The more development work that can be shared in this way, the more attractive the native open-source path becomes, even from an economic perspective. Community code alone does not put vehicles on the road. The open core ends at a clearly identifiable point, and the commercial service begins from there:

- Certification to ISO 26262. The evidence itself is already generated during the open process and is available to all participants. The assurance that the program is robust, the certificate issued in a corporate context and maintenance through-out the program's lifetime are purchased

- Long-term support over 10 to 15 years
- Contractually defined response and resolution times for safety- and security-related defects based on the version currently in series production
- Platform validation for the specific combination of operating system, computing platform, and toolchain
- Commercial liability for a specific release version, i.e. the contractual assumption of risk by a company
- Ongoing participation in the project's governance bodies, which is a prerequisite for remaining in the mainline

These services are required in every model and cannot be provided by a community. It cannot guarantee a production version for three years, nor can it operate a validated hardware laboratory or accept any liability. The difference compared to the proprietary model lies in transparency: because the code is public, the service constitutes the entire subject matter of the contract and can therefore be assessed on a case-by-case basis. Payment is made for risk reduction, speed, and reliability.

Two points that belong in every contract

The open-source model presents two areas of concern for procurement. Both need to be discussed.

First, development costs must be covered upfront. Published code cannot be billed retroactively. A platform of this scale can hardly be fully pre-financed and later

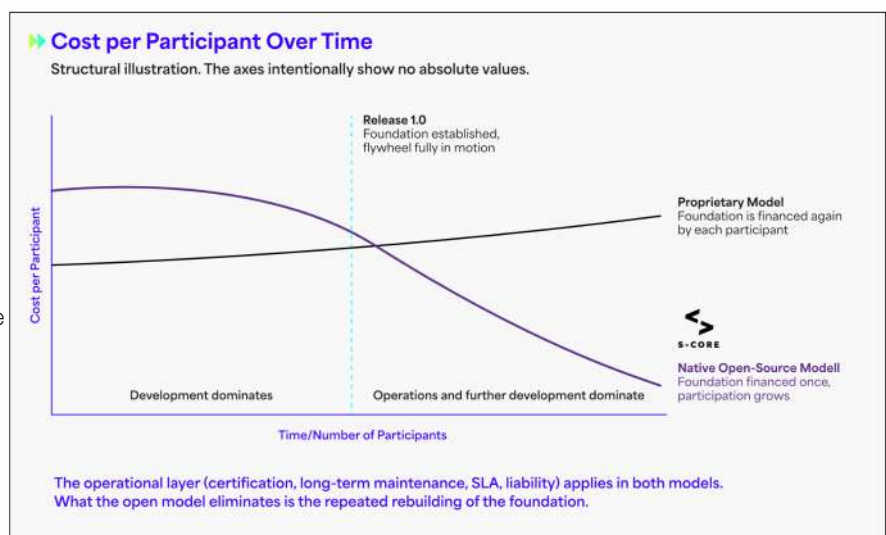


Figure 4: Cost per participant over time – proprietary model versus native open-source model. Structural illustration without absolute values. © Qorix GmbH

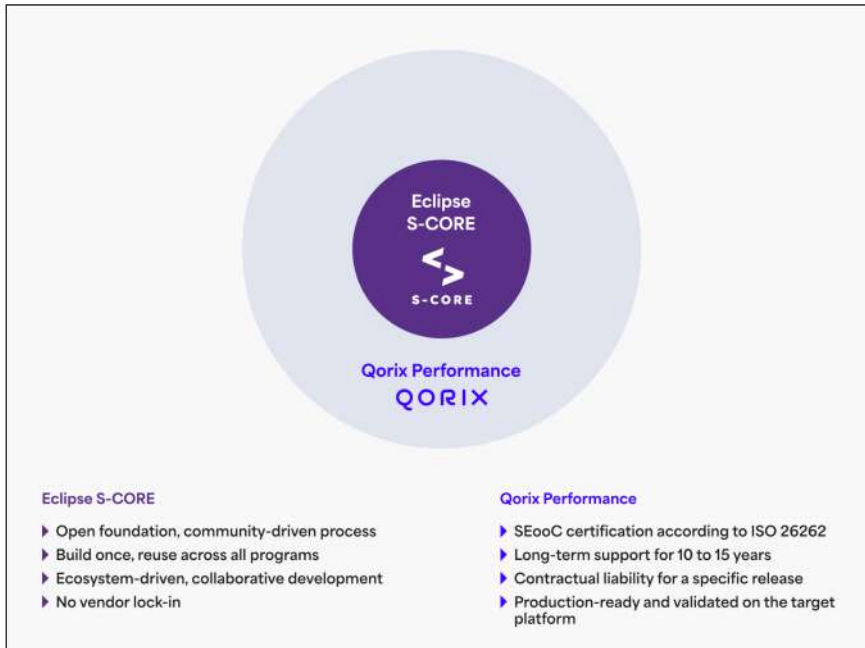


Figure 5: Same code, commercial packaging: Qorix Performance transforms Eclipse S-CORE into a commercially supported, production-ready distribution, complete with certification, long-term support, and contractual liability. © Qorix GmbH

refinanced through licenses alone. Therefore, the path to the first viable version appears more expensive than purchasing a ready-made, proprietary solution. Those who provide early funding help determine the scope and priorities. They remain unbound, even during this phase, and gain predictability regarding a fixed production launch date. Those who wait may end up getting the foundation at a lower cost.

The key factor is the distance from the upstream repository. A distribution can start with just a few patches. However, the patch stack grows with every release and special case. Without discipline to continuously upstream them, critical functionality will eventually be exclusively in the patch stack, and lock-in will return – just in a different form. With the first patch, this risk remains invisible. It becomes apparent years later in the development and integration efforts, and ultimately in the costs of switching. Therefore, the contract must include the discipline of upstreaming patches, with the burden of proof for any deviation resting with the provider.

The journey from open source to a production-ready product

Qorix Performance is a commercially supported, production-ready distribution

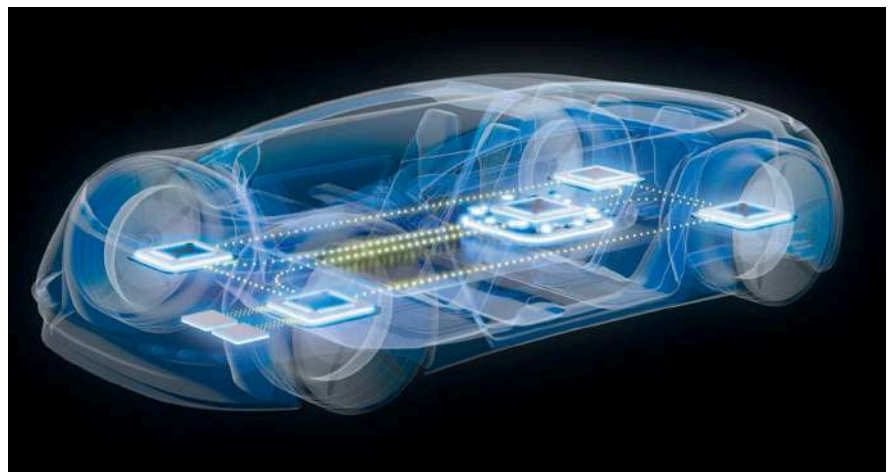


Figure 6: Qorix Performance turns Eclipse S-CORE into safe, production-ready middleware featuring determinism, zero-copy communication, mixed-criticality orchestration, and real-time behavior. © Qorix GmbH

of Eclipse S-CORE. It is designed for high-performance and central compute ECUs as well as for ADAS and AD applications (**Figure 5**). Development takes place entirely in the public S-CORE repository. There are no private forks or proprietary patches, and the gap to upstream is zero by design. Applications are developed directly against the S-CORE interfaces.

The offering fulfills two functions. First, it provides certification, contractual commitments, and liability for a production program. It provides ASIL-B certification artifacts as a Safety Ele-

ment out of Context, validation on the target platform, defined releases with long-term support, and response times tailored to the production requirements. Second, it enables the targeted closure of gaps in the open core and the further development of the common foundation.

The two functions are interrelated. The commercial contract funds work on the core, which is subsequently made available to the entire industry. The goal is to create a fully open core for automotive software, because a shared foundation allows development effort to be shared across company boundaries, which keeps the market open to all vendors.

The upcoming Eclipse S-CORE Release 1.0 is often considered as a finish line. For a program, however, it is just the beginning. Once the foundation is established, the effort shifts to operation and further development, and the circle of those who can build

upon it grows significantly. From this point on, the flywheel effect benefits the broader industry, extending beyond the initial participants. For OEMs and vendors, this raises the question of how much development work they will need to contribute to the foundation in the future and where their vehicle software will actually begin to differ.

■ (ah)
www.qorix.ai



Lars Bauhofer is Product Program Manager at Qorix GmbH.
© Qorix GmbH