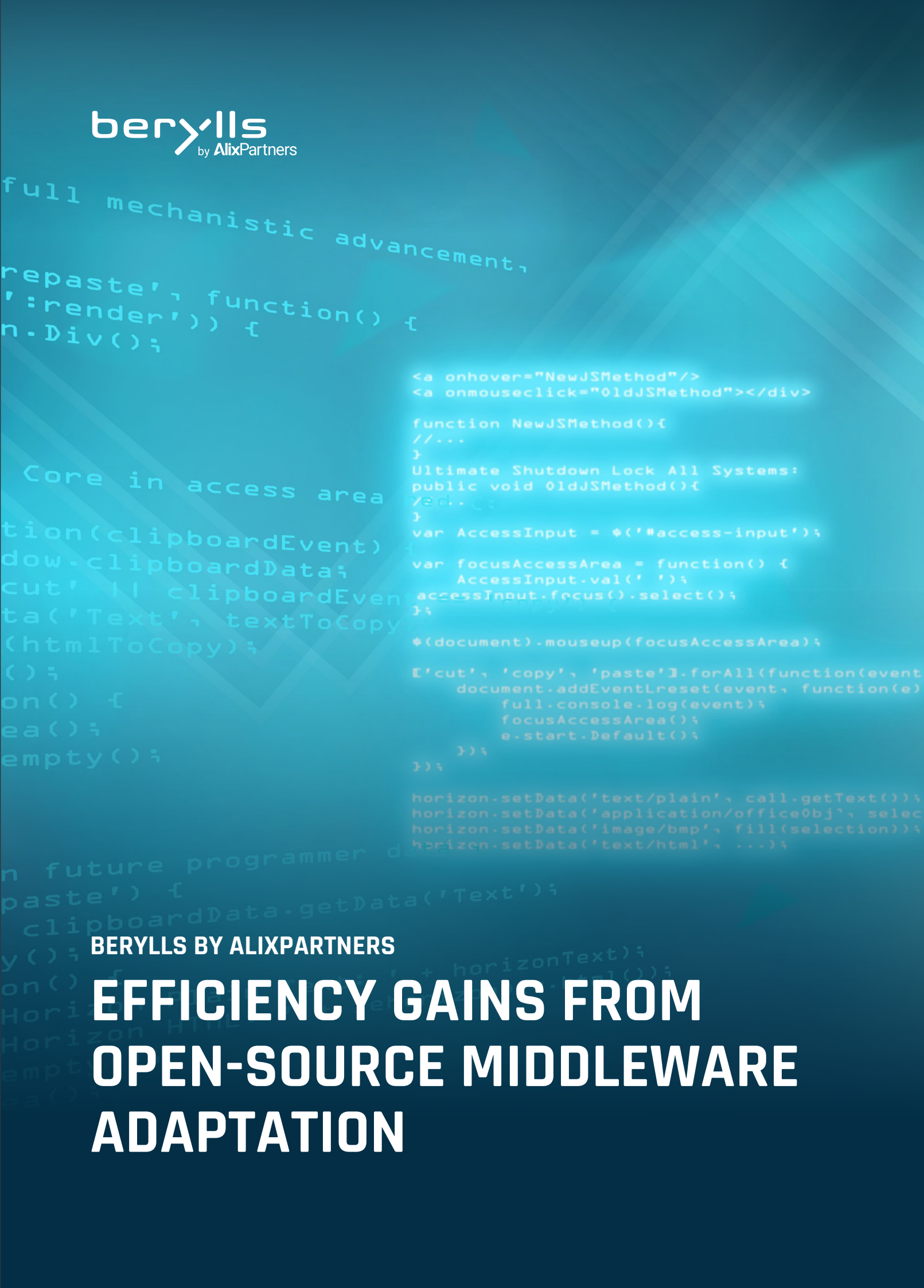

BERYLLS BY ALIXPARTNERS

EFFICIENCY GAINS FROM OPEN-SOURCE MIDDLEWARE ADAPTATION

Executive Summary

Open-Source middleware in automotive has moved past the proof-of-concept stage. The Eclipse Foundation's S-CORE project turns Original Equipment Manufacturer (OEM) adoption from a hypothetical into a concrete decision within the next 24–36 months. This paper evaluates the strategic case for participation from the perspective of R&D leaders, across three OEM and three supplier archetypes.

Open-source middleware should not be considered a universal-, but rather a strategic control lever. Its primary value is not free software, but reduced dependency, accelerated standardization, and reallocation of engineering effort away from non-differentiating layers.

Across the four hypotheses, the verdict is differentiated, not uniform. On lifecycle effort, the open-source path delivers up to ~20–40% lower one-time costs for contributing OEMs through shared joint development across a minimum of five contributing parties (OEMs and contributing suppliers) and the strong reduction of per-vehicle license fees. Even higher savings can apply if an existing solution without development contribution can be used. On time savings, Software OEMs with high in-house software competences see roughly neutral outcomes (in-house baselines already short). Legacy OEMs in SDV Transformation and Technology Laggers save ~10–20% (three to ten months time) however, the latter need suppliers to be able to realize this advantage due to poor internal software skills. On product-lifecycle stability, the open-source solution matches proprietary middleware once the safety case is built and sufficient testing has been carried out (multiple applications from OEMs). On lifecycle risk, exposures shift from project-specific technical debt to shared governance dependence, leaving total risk no higher than proprietary development.

Winners of this transition are Legacy OEMs that shape the standard early as well as Traditional System/ Middleware suppliers that reposition from license vendors to integration and adaptation partners. Technology Laggers are offered the chance to catch up to advanced competitors at lower cost by utilizing such suppliers. Scaling of SoC Suppliers and Hyperscalers is further strengthened from their current position. Artificial intelligence (AI) bears on this shift both as a tool that accelerates middleware development and adaptation and as a workload the middleware must carry increasingly. The quantitative model here holds AI tooling constant and leaves those effects out of scope.

1 | Intro

Open-source software has reshaped consumer computing, enterprise IT, and most layers of the cloud stack. Automotive has adopted it selectively, mostly at the operating-system layer through Linux in in-vehicle infotainment. The next layer up, middleware, has stayed largely proprietary and concentrated in a small set of Tier 1 suppliers. Industry positions on software efficiency frame open-source middleware as the next source of cost and time savings in software-defined vehicles (SDV).

Open-source middleware shifts the locus of competition away from the non-differentiating middleware layer itself, toward the speed and quality of adaptation on top of a shared baseline. Adaptation speed, not license cost, is where both the time advantage and the strategic exposure sit.

This paper evaluates that question on strategic advantage of Open-source middleware in relation to efficiency gains along four hypotheses: total cost over lifecycle (H1), combined development and adaptation time (H2), middleware stability across the product life cycle (H3), and overall risk exposure (H4). Each hypothesis is evaluated against three OEM and Supplier archetypes. The archetype split matters due to fundamentally different starting positions, strengths and weaknesses as well as business models behind the actors that result not only in varying efficiency gains but also in different risks that apply.

One boundary deserves explicit mention. Artificial intelligence (AI) intersects open-source middleware along several axes. AI-assisted coding compresses the development and adaptation effort quantified in this paper even further. AI is beginning to automate the verification, validation, and safety-case work that drives stability and risk and as application-layer code becomes cheaper to generate, the focus of differentiation may shift. These effects have an impact on the numbers in this paper. The quantitative model deliberately holds AI tooling constant and out of scope, so these effects sit outside the present scope and are left to further analysis.

2 | General Understanding

2.1 Automotive Software Stack and the Role of Middleware

Vehicles today contain dozens of electronic control units (ECUs), each with different performance levels, safety requirements, and sometimes life cycles. To manage this complexity, automotive software is organized into clearly defined layers.

At the bottom of the stack is hardware: processors, memory, sensors, actuators, and communication interfaces. Directly above it sits the base software, including the operating system (OS). It ensures software runs reliably on specific hardware.

Above the OS lies middleware. Middleware provides the mechanisms that allow software components to interact – within the same ECU or across the entire vehicle. In simple terms:

- **The OS decides how software runs.**
- **Middleware decides how software components talk to each other.**

Middleware deliberately avoids vehicle-specific behavior. It provides generic infrastructure – communication, lifecycle handling, diagnostics, data exchange, and basic security. Because these functions are broadly similar across Original Equipment Manufacturers (OEMs), middleware is not a point of competitive differentiation.

Above middleware, application frameworks incorporate domain knowledge and OEM-specific conventions. This is where vehicle behavior becomes differentiating.

The boundary between middleware and application frameworks is defined through standardized, platform-agnostic APIs. These interfaces make it irrelevant for developers whether communication is local or remote, which network is used, or on which hardware the service runs. Standardization at this layer enables reuse across platforms, simplifies integration, and supports long-term maintainability.

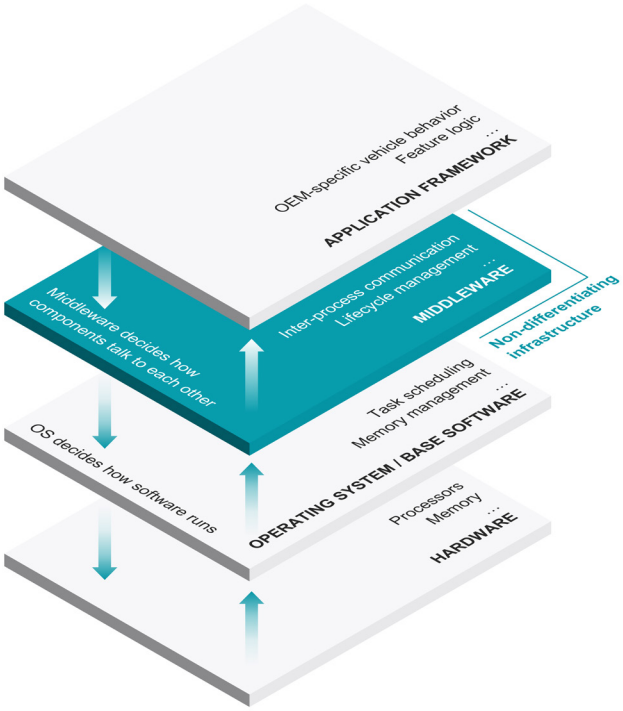

Figure 1. Automotive software stack – layers and boundaries

2.2 Open-Source Software and its role in automotive

Open-source software has made a strong entry into automotive primarily through Linux in in-vehicle infotainment (IVI). Beyond the operating system layer, open-source middleware is the next dimension, exemplified by initiatives such as Eclipse S-CORE (Safe Open Vehicle Core).

Open-source middleware in automotive forms either through a single dominant actor (e.g., Google Android Automotive) or through organized communities (e.g., Eclipse S-CORE, Automotive Grade Linux). In the community model, OEMs and suppliers jointly shape architecture and requirements and contribute individual building blocks – either from existing solutions or as new code.

OEMs typically contribute to reduce vendor lock-in, standardize platforms, and lower the total cost of non-differentiating middleware across brands and programs. Tier 1/2 suppliers often upstream components that they also offer as hardened, supported products, earning revenue from integration, customization, and long-term maintenance rather than from the OSS artifact itself. Suppliers and Hyperscalers position their technologies as de facto SDV standards, monetizing through tools, platforms, and managed or cloud services.

Most OSS initiatives in automotive avoid defining a common commercial model; they focus on technical scope, governance, and neutrality, leaving business models to ecosystem participants. This separation maintains antitrust compliance, preserves trust among competitors, and allows diverse monetization strategies to coexist on top of the shared OSS base.

Open-source components must integrate into ASPICE-conform development and verification workflows. The fulfilment of UNECE cybersecurity and software-update regulations, functional safety (ISO 26262) and ISO/SAE 21434 for series deployment is also required

2.3 Effects for OEMs and Suppliers

Open-source software has made a strong entry into automotive primarily through Linux in in-vehicle infotainment The adoption of open-source middleware fundamentally changes automotive software development for both OEMs and suppliers. Open-source may suggest immediate cost and time advantages by avoiding license fees and reducing proprietary development; the reality is more differentiated.

Cost Structure

Cost drivers run along the product lifecycle: development, adaptation, and operation. The OSS path adds one step, hardening of components, to this structure. Development and adaptation are one-time efforts and do not scale with vehicle volume; operations scales with vehicles through manufacturing costs that include in-house personnel, external services, infrastructure, tooling, and license fees. The differentiating factor across development and adaptation is the OEM's share of internal capability versus outsourcing.

Impact on OEMs

Three OEM archetypes emerge from five characteristics relevant to the analysis.

	Software OEM	Legacy OEM in SdV Transformation	Technology Lagger
Value creation	High share of In-house development (>50%)	Partly In-house development (~20%)	Almost no In-house development (<5%)
Production volume	Low-Mid Volume	High Volume	Mid-High Volume
Developer count and skill level	Plenty talented developers	Some talented developers	Few less qualified developers
IT Infrastructure	Modern toolchain & backbone	Partly Legacy toolchain	Legacy toolchain
Development time for middleware	30-36 Months	42-48 months	45-52 months

Figure 2. OEM Archetypes – characterization

A Software OEM brings high software expertise and comparatively low production volume; co-developing modular middleware sits naturally inside that profile. The main question for such an OEM lies in the trade-off between efficiency gains for itself and the potentially higher gains for competitors due to contributed knowledge.

Legacy OEMs in SDV Transformation are sufficiently experienced to contribute to such an open-source middleware and to capture the resulting efficiency gains. The magnitude of the effect depends primarily on how broadly the development effort is shared across co-developing OEMs and suppliers, and on the standardization gains realized through aligned toolchains.

Technology Laggers lack the in-house software capability to make a meaningful contribution to an open-source middleware. They can therefore only wait for other competitors to deliver the open-source baseline and then capture efficiency gains via supplier-led adaptation to their own stack.

Challenges for Suppliers

The main role for suppliers in a traditional setup was the development and maintenance of software, including middleware as well as the hardware production and cloud backend. This business model is partly disrupted by the availability of Open-Source Middleware. On the supplier side, a distinction can be made between three different groups, the Traditional System/Middleware Suppliers, Hyperscalers and SoC Suppliers.

Traditional System/Middleware Suppliers build and sell full hardware-software systems; an available OSS baseline displaces that model and forces a repositioning in the value chain.

Due to the Hyperscalers' offering on cloud backends, a standardization of middleware mainly simplifies the OEM specific interfaces, resulting in even simpler scaling of their solutions. The challenge lies within the process of adjusting their product interfaces to the new standards.

Typical SoC Suppliers do not offer middleware solutions but rather basic driver software. The hardware agnostic middleware on top does not directly interfere with their business model. Newer players like NVIDIA, however, are expanding their portfolio and offer integrated stacks and broader development toolsets which potentially conflict with open-source environments. For such companies, the attractiveness of their stack is challenged by open-source solutions. Whether the platform offers enough unique benefit to justify its price and lock-in is the open commercial question for these suppliers.

3 | Hypotheses

The claim that open-source middleware is advantageous relative to proprietary solutions in terms of efficiency can be evaluated based on four working hypotheses.

3.1 The total effort in one-time expenses and manufacturing costs (software licences) for open-source middleware over its lifecycle is up to 40% lower than for proprietary in-house development.

Across all three phases, the effects of open-source middleware differ between OEM archetypes and suppliers.

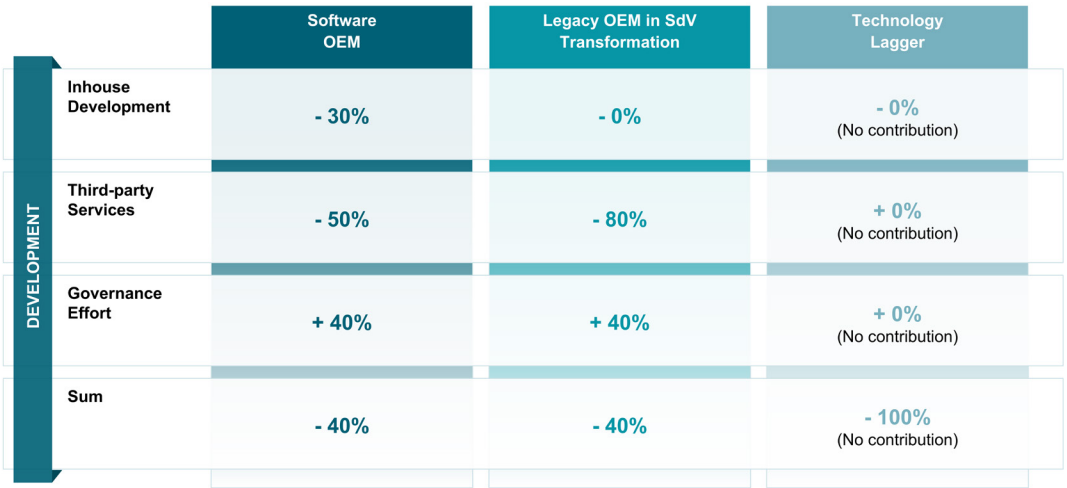

Figure 3. Effects on OEM Archetypes during Development Phase

In the development phase, the base assumption that open-source development profits from shared contributor effort holds true. Equal contribution of five parties (OEMs and participating suppliers) is assumed, cutting per-OEM development effort to 20% (versus pure in-house) for Software OEMs and Legacy OEMs in SDV Transformation. Technology Laggers lack the knowledge and capacity to contribute, so they take the OSS baseline without contributing, eliminating development cost entirely; governance overhead therefore applies only to the two contributing archetypes.

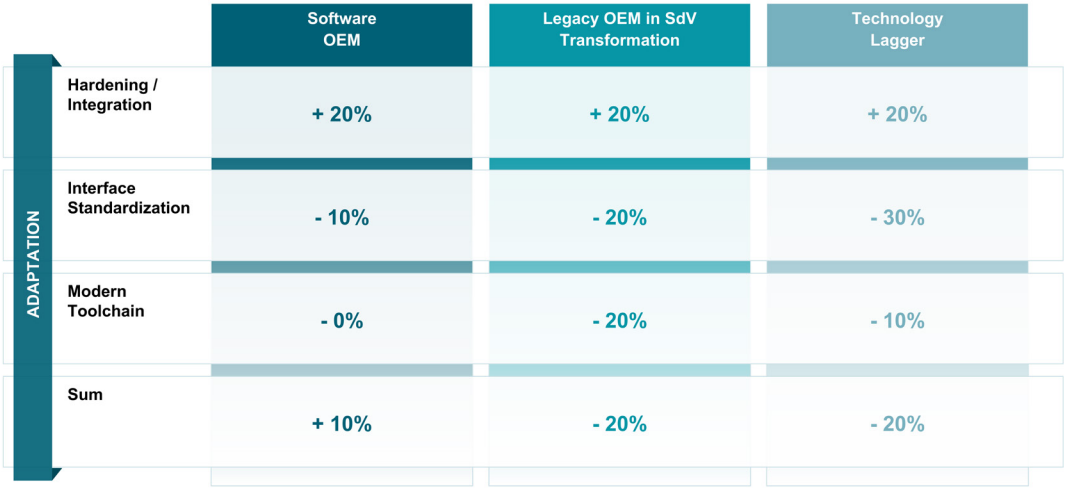

Figure 4. Effects on OEM Archetypes during Adaptation Phase

Adapting open-source middleware adds approximately 20% effort across all archetypes through hardening. For Technology Laggers this work is performed by suppliers, given limited in-house middleware knowledge. Counteracting effects come from modern toolchains and high interface standardization, which Software OEMs already exploit.

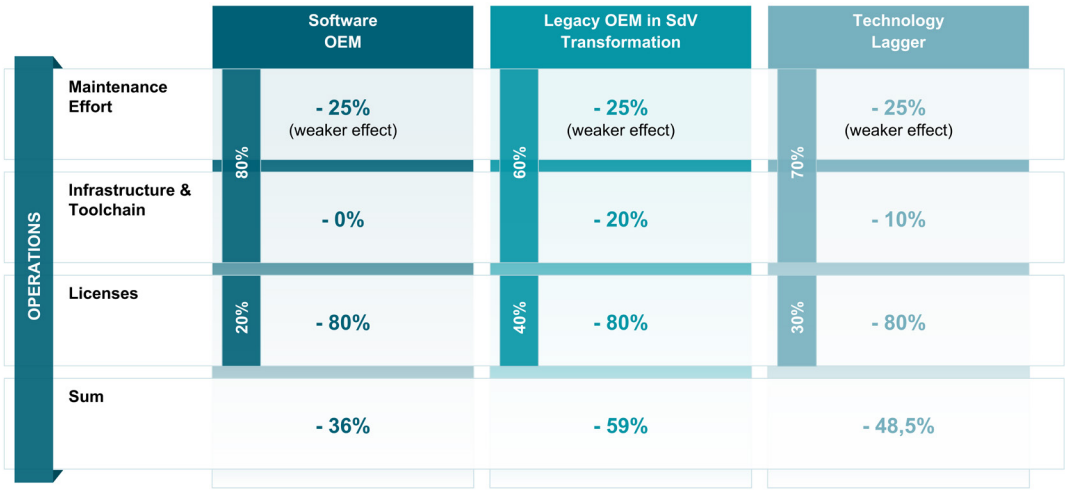

Figure 5. Effects on OEM Archetypes during Operations Phase

During operations the cost logic changes because license costs scale with production volume. Software OEMs typically have lower volumes, so licensing accounts for a smaller share of total expenses. The opposite holds for the other archetypes. A second factor is fixing field defects that are partly OEM-specific and not handled by the community alone, applying the development-phase effect in weaker form. Toolchain savings apply as in the adaptation phase.

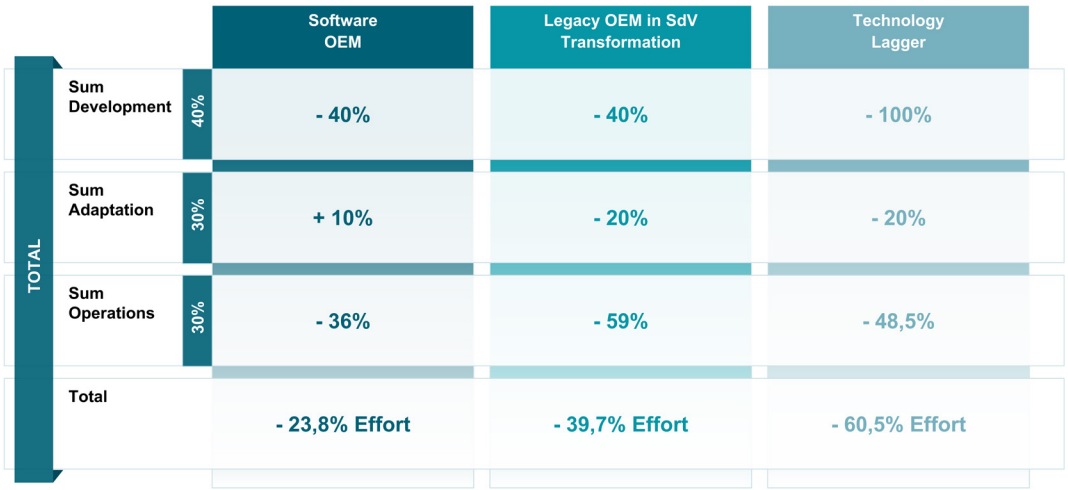

Figure 6. Total effects on OEM Archetypes

Across the three phases, Software OEMs see low effort saving, since the in-house development is already efficient and Open-Source Middleware adds coordination and hardening overhead. Software OEMs may still participate for standard-shaping reasons. Legacy OEMs in SDV Transformation capture meaningful benefit, depending on the number of contributors and equal distribution of effort. Technology Laggers capture meaningful savings through supplier-mediated access, with the magnitude conditional on whether their supplier participated in the community-

Effects for suppliers vary by archetype. Traditional System/Middleware Suppliers face a participation incentive through standardization and knowledge gain, set against the disincentive that contributing code accelerates the loss of their proprietary middleware IP. On the effort side, their shared development mirrors the Legacy OEM in SDV Transformation profile. The adaptation phase opens a major opportunity for the same archetype: Legacy OEMs in SDV Transformation need integration support, Technology Laggers are entirely dependent on it, and playing the integrator role may be the only viable survival path. Hyperscalers see limited open-source middleware impact,

primarily via scaling benefits from standardized interfaces across cloud customers. Efforts are limited to adjusting these interfaces to newly set standards and governance activities in case of participation within the open-source community. SoC Suppliers face the same pattern, conditional on not pursuing a proprietary ecosystem strategy, the contribution in terms of own SoC support can benefit the customer base at cost of governance efforts.

A cost analysis across the three phases cannot be generalized for any of the three supplier archetypes, as it depends heavily on the underlying strategy and business model. The shift from development services to adaptation and maintenance means the loss of one of their business areas for many suppliers. On the other hand, it also presents an opportunity to tap into new business models ahead of all competitors through OEM partnerships and dedicated positioning in the open-source market. Therefore, H1 cannot be verified for the supplier side.

3.2 The combined development and adaptation time of open-source middleware is up to 20% lower than a comparable proprietary development.

The model compares two scenarios to deployment-ready middleware. In the in-house scenario, a single OEM develops and adapts the stack independently. In the open-source scenario, multiple OEMs jointly develop the stack, after which each OEM individually adapts the result for its own production context. All open-source adopters base their work on fully developed middleware.

Phase 1: Development

The baseline for in-house development is set by current average middleware development times, considered independently of vehicle development projects and their duration.

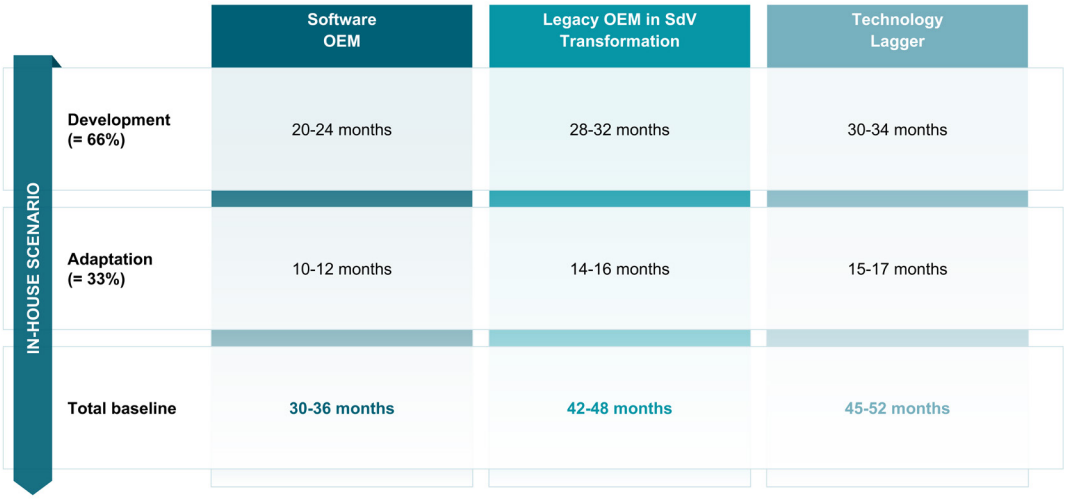

Figure 7. Counterfactual baselines: total in-house scenario by archetype

The total development duration (development + adaptation) for the in-house scenario can be seen as a working assumption, though the 30–36 month range for Software OEMs is close to observed timelines.

As stated earlier, the joint development assumes approximately five contributors with about equal contribution. Current open-source data does not fully reflect this assumption (e.g., S-CORE). The defined and communicated timeline of S-CORE however sets the pure development time without adaptation to 24 months. Of those joint development months, roughly 7 months are absorbed by coordination overhead (estimated at 40%) and roughly 17 months by productive development. The 17-month productive figure is comparable to what a single Software OEM would need to develop equivalent middleware alone. This is possible because multiple contributors develop in parallel and reuse existing modules as a baseline. The 40% coordination figure is the most sensitive single assumption in the calculation: at 50%, joint development would extend to over 25 months and all downstream savings would compress proportionally.

Joint development at 24 months is comparable to a Software OEM's solo speed (20–24 months), substantially shorter than a Legacy OEM in SDV Transformation (28–32 months), and shorter than supplier-assisted development for a Technology Lagger (30–34 months). For the latter two archetypes, the development phase is where the calendar-

time saving is generated; for the Software OEM, the development phase delivers no calendar advantage, and the value of joint development lies in shared effort and standard-shaping rather than shared time.

Phase 2: Adaptation

Adaptation covers the work required to make the jointly developed stack production-ready for a specific OEM context. This includes hardening, safety qualification, OEM-specific integration, and performance tuning. All three archetypes adapt and harden the OSS middleware to their own architecture and internal quality and safety standards.

To maintain the two-thirds/one-third relationship between development and adaptation, the adaptation baseline is derived from the joint development timeline. A development phase of 24 months implies a base adaptation time of approximately 12 months. This represents the raw adaptation effort before any hardening or OEM archetype-specific adjustments.

Hardening the OSS codebase to OEM-specific quality and safety standards adds roughly 20% effort on the joint adaptation base translating into approximately 20% more calendar time. Archetype-specific factors then deviate from this baseline.

The hardening overhead applies in full to Legacy OEMs in SDV Transformation, setting their adaptation time to ~14 months. Legacy OEMs that contributed know the OSS middleware's architecture and interfaces and their own vehicle architecture, but their tool chains are older, their teams are less experienced with OSS integration patterns, and their processes are less optimized for externally governed code.

Software OEMs adapt faster than the baseline. These OEMs possess deep knowledge of their own vehicle architecture, operate modern development tool chains already compatible with the OSS build infrastructure and employ developers experienced in integrating external open-source components. The net effect is an OSS adaptation of approximately 12 months.

Technology Laggers adapt slower than the baseline, with adaptation performed entirely by Traditional System/Middleware Suppliers. A critical premise is that the supplier itself participated in joint development, so it knows the middleware: its architecture, interfaces, conventions, and quality standards. What it does not know is the specific vehicle architecture it is adapting the middleware for. This architecture-specific learning, paired with additional coordination between supplier and OEM, is the primary source of the additional overhead. The net effect is an open-source middleware adaptation of approximately 18 months.

This creates a direct strategic incentive across the supplier triad. OEMs adapting OSS middleware will prefer Traditional System/Middleware Suppliers that participated in the open-source development, because those participants bring middleware-internal expertise that non-participating suppliers cannot match. Hyperscalers and SoC Suppliers occupy a different position: they do not act as integrators of the middleware itself but reposition along adjacent layers of the stack.

Combined Results

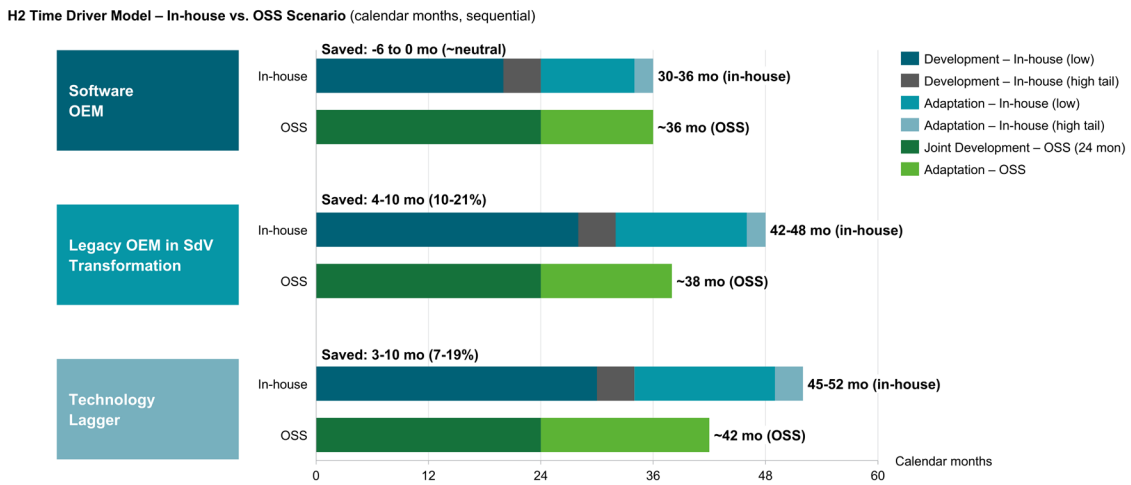

Figure 8. Time Driver Model – in-house vs. OSS development (calendar months, sequential assumption)

Software OEMs see a roughly neutral result: joint development matches their solo speed (20–24 months) and adaptation runs close to in-house pace (~12 vs. 10–12 months). Time savings are therefore not a compelling reason to adopt open-source middleware.

Legacy OEMs in SDV Transformation save roughly 4–10 months calendar time, driven by the development phase; adaptation runs faster than in-house pace because the 20% hardening overhead on the joint baseline is smaller than the legacy organization's solo adaptation effort. Percentages compare the Open-source scenario against each end of the in-house range: 10% saving versus a 42-month in-house development and adaptation, 21% saving versus a 48-month one; the same convention applies to the Technology Lagger figures.

Technology Laggers do not contribute to joint development. Counterfactual is supplier-assisted proprietary development and adaptation (45–52 months). The potential savings in this scenario are roughly 3–10 months of calendar time (7–19%). The saving comes from compressed development; adaptation runs longer due to the architecture-specific learning carried by the supplier. The primary value is technology access at lower total cost.

Effect of Early Engagement

The calculation treats development and adaptation as strictly sequential. This is a conservative simplification. In practice, OEMs can begin adaptation work before V1.0, using intermediate releases and architecture previews. Architecture mapping, toolchain setup, team training, and proof-of-concept deployments on target hardware can all proceed before the final release, increasing the time-saving further.

Implications for Suppliers

The supplier-side impact differs sharply across the three supplier archetypes.

Traditional System/Middleware Suppliers carry development time analogous to the OEM benefit. Contributing to a 24-month joint development cuts middleware development effort by roughly 40% relative to a standalone build, while reducing per-customer adaptation overhead by leaning on the same hardened OSS baseline OEMs use. In the adaptation phase they typically act as integrators on behalf of OEMs, and the middleware-internal expertise built up during contribution pays off through faster, lower-cost integration. This also reflects a shift in these suppliers' business models, moving away from development toward integration and adaptation.

Hyperscalers' cloud backend software does not contribute to the in-vehicle middleware itself; engagement amounts to roughly 10% coordination overhead on a small platform-alignment base, mainly to align proprietary cloud-protocol interfaces with the OSS standards emerging from the community.

SoC Suppliers face the same approximate time numeric as Hyperscalers. Adaptation time scales favorably across customers under standardized SoC-middleware interfaces, with the caveat that SoC Suppliers pursuing their own proprietary middleware stack on top of the silicon erode the scaling benefit.

3.3 "Open-Source Middleware is as stable as a proprietary solution throughout the product life cycle."

Stability in middleware must be assessed across three dimensions: architecture stability, API and interface stability, and operational stability. In an open-source context, these dimensions are tightly connected. Architectural decisions shape interface behavior, and both influence long-term behavior in operations. Open-source governance becomes a structural enabler of stability rather than an administrative afterthought.

Architecture Stability

Architecture stability depends on how well the middleware absorbs change without forcing disproportionate redesign. The main drivers are inter-module dependencies, hardware-related design constraints, and the rate at which requirements change. Modularity is the central design choice: minimizing tight coupling and favoring well-defined interfaces lets the system withstand externally driven change without rewiring the core.

In practice, open-source middleware should pursue an aggressively modular design following clear automotive standards that preserves flexibility beyond the immediate baseline. Its effort distributes across contributors and work packages, making modularity both a design principle and a structural advantage of open-source development.

API and Interface Stability

API and interface stability is the most governance-sensitive dimension. The core risk in open-source initiatives is uncontrolled change: if interfaces can be modified too freely by individual actors or OEM-specific branches without disciplined reintegration, fragmentation becomes a realistic threat. Open-source governance models exist precisely to define authority, process, and accountability for such changes.

A strict governance model with clear approval and change processes is required to coordinate a broad contributor community: formal review paths, explicit ownership, and consistent acceptance criteria for interface changes. Maintainer responsibility must be assigned to dedicated, recognized teams who carry decision authority over the affected interfaces.

Operational Stability

Operational stability describes how reliably the middleware behaves in real deployment conditions over time. In the early stages, open-source middleware may be less stable than a mature proprietary alternative due to a converging contributor base, limited deployment diversity, and uneven validation depth across use cases. This is a typical maturity effect: as use cases multiply and the user base broadens, open-source middleware can reach operational stability quickly and surpass single-vendor stacks driven by the broader defect-discovery surface.

Implications for OEMs and Suppliers

The most direct mechanism for ensuring stability is governance and active steering. Architecture can be designed for resilience, interfaces can be versioned correctly, but without disciplined coordination the benefits of openness can easily be offset by fragmentation, inconsistent decision-making, or delayed reintegration of operational findings. Governance is part of the technical system design, not project-management overhead.

For OEMs that want to reduce stability risk and suppliers that are looking for alternative business opportunities, the rational strategy is to invest directly in coordination mechanisms and to participate actively in the governing structure of the middleware initiative. This includes funding or staffing maintainer roles, enforcing formal change processes, shaping interface policies, and ensuring that field data flows back into the shared code base. Such a role is best filled and sustained over the long term by Traditional System/Middleware Suppliers. Hyperscalers and SoC Suppliers do not carry direct in-vehicle stability obligation, but interface stability.

Subject to governance investment and user scaling, H3 holds: OSS can match proprietary stability, but not automatically

3.4 "The total risk of an Open-Source Middleware over the product lifecycle is no greater than that of a proprietary in-house development."

Risk in proprietary middleware concentrates on one supplier, one stack, and one decision node. Risk in an OSS path distributes across the community, the integrator, and the contributor community. Neither topology is risk-free. The question is how the risk profile differs and if an overall shift towards one or the other solution can be observed. The comparison runs across five dimensions, for OEM and Suppliers archetype by archetype.

Dimension 1: Supply-Chain and Dependency Risk

Vendor lock-in dominates the economic exposure. Re-integration and re-certification costs after a stack change deter most mid-program switches, and the supplier prices its captive position into every renewal – each additional vehicle program built against the stack raises the cost of leaving it. Proprietary middleware is typically a single-source dependency, which leaves the OEM exposed to supplier outages, insolvency, and ownership changes that may alter the commercial or regulatory terms of the relationship.

Open-source middleware reduces supply-chain risk through breadth of contribution. The OEM depends on the community, on participating integrators, and on the broader contributor community rather than on a single vendor. Each individual exposure is smaller and partially substitutable if the overall community and organization hold. If members cannot maintain consensus on architectural direction they may leave and the effort-sharing logic of OSS collapses. The refocus of the GENIVI Alliance, an automotive Linux community founded in 2009 whose assets transferred to the Connected Vehicle Systems Alliance (COVESA) in 2021, shows that automotive communities can lose viability after a decade of operation. The risk of such a total failure is comparable to having an insolvent supplier in a proprietary scenario. The first requires the OEM himself to step in and maintain the software whereas the latter bears high cost of supplier replacement for development and maintenance.

For Technology Laggers specifically, the dependency and associated risk of open-source middleware is higher than for the other two archetypes of OEMs. Due to missing capabilities to integrate themselves, the availability and continuous support of a system/ middleware supplier is needed. If this supplier departs with the open-source community, changes and bugfixes might not be handled as quickly and effectively as required.

Dimension 2: Technical and Safety/Security Risk

Functional safety is the responsibility of OEMs. The type-approval holder constructs the safety case for the integrated system regardless of stack origin. In both scenarios the middleware must be qualified to the required ASIL level; the obligation does not relocate to the supplier or to the community. Depending on the pre-qualification of open-source middleware, this could lead to higher risk for OEMs. The S-CORE project addresses this risk by among others building on QNX OS for Safety microkernel (SDP 8.0) as a pre-qualified safety foundation.

Cybersecurity exposure reverses across the two scenarios as stated earlier. Proprietary code is closed and harder to attack, but vulnerability can only be remediated by the supplier and remediation cycles run on the supplier's release calendar. Open-source code is public and easier to attack, but patches can be authored by any qualified contributor and reach the OEM faster. The trade-off is real and the net level is at parity between both scenarios.

Dimension 3: Strategic and Organizational Risk

Community governance is the OSS-specific risk class without a proprietary mirror. Two failure modes apply. The first is total community failure as described in Dimension 1. The second, and more frequent, is being overruled. The OEM's preferred architectural direction is voted down by other members. This is particularly material in the communities' early years, when architecturally critical decisions on interface design, safety partitioning, and tooling are taken with long-horizon consequences and no individual member holds a controlling vote.

Besides tooling decisions, development processes bear an organizational risk. Most OEMs and Tier 1 suppliers work to the V-Modell, a gate-based development process aligned to long product cycles and bilateral specification handoffs. Open-source middleware moves at a different pace (continuous, merge-driven, distributed across many contributors). Processes and organization must adapt to the cadence. During the transition, delivery risk is elevated because the new governance and operating model do not yet hold. The risk decreases as soon as the OEMs / supplier operating model is in line with open-source development.

Dimension 4: Regulatory and Product-Liability Risk

Both scenarios produce regulatory and product-liability risk on the OEM. UN R155 (Cybersecurity Management System, CSMS), UN R156 (Software Update Management System, SUMS) and the General Data Protection Regulation (GDPR) to name just a few cause obligations with the type-approval holder regardless of stack origin.

For proprietary middleware, warranty terms cap supplier indemnification but provide a defined claim mechanism against which financial exposure can be priced. In the OSS path no contractual chain to the open-source community exists; contributors transfer code under permissive licenses without ongoing liability. The OEM could purchase comparable cover separately at the integration layer: the integrator or distributor absorbs the contractual exposure that the community does not carry, priced as a risk-transfer service.

Dimension 5: Middleware Quality and Maturity

Middleware quality risk distinguishes the two scenarios along an evolution curve rather than at a fixed point. Many adopters produce many integration paths, which produce a wider defect-discovery surface than any single vendor can match internally. The curve is therefore not a one-time comparison but rather a trajectory.

In the early phase of community output, open-source middleware carries higher quality risk than a mature proprietary stack. The code base is less battle-tested in operation, and the OEM is among the first to encounter integration edge cases. Proprietary middleware deployed across multiple vehicle programs for a decade has accumulated field-defect repair and benefits from a defect-discovery base that an early-stage open-source middleware does not yet have.

The curve crosses. As the community output is deployed across multiple OEMs and integrators, the user base widens, defect-discovery rate accelerates, and quality risk falls below the proprietary baseline. Single-vendor proprietary middleware has a quality ceiling bounded by one organization's QA capacity and by the volume of programs its customers run. OSS middleware has no equivalent ceiling: quality continues to improve with each additional community adopter, and the marginal cost of defect discovery is shared across the user base rather than absorbed by one supplier. To hold true, it is essential to re-integrate fixes on OEM specific adaptations of the middleware back to the open-source baseline.

Archetype-Specific Synthesis

The five risk clusters resolve to different net positions across all OEM and Supplier archetypes:

- **Software** OEM: total risk roughly comparable to the proprietary path, slightly higher; OSS-specific exposures in community governance and pre-crossover quality dominate over modest time and effort savings.
- **Legacy OEM in SDV Transformation:** slightly lower total risk. The dominant effect is a risk shift from supplier dependency and capped contractual recourse toward community governance and integrator-contracted recourse.
- **Technology Lagger:** slightly lower total risk, dependent on the specific OEM–supplier adaptation setup. The integrator's own community position determines how much of the dependency and quality benefit reaches the OEM. The dominant effect is again a risk shift.
- **Traditional System/Middleware Supplier:** higher total risk, depending on the positioning as integrator / maintainer for Legacy OEMs in SDV Transformation and Technology Laggers. The business model flips from licensing toward integration and safety-case engineering services, the transition window of three to five years is the moment of highest fragility. Failure to adapt leaves the supplier in a materially weaker market position than today.
- **Hyperscaler:** lower risk. The main risk is in commoditization of proprietary cloud protocols and adjusting interfaces to open-source standards.
- **SoC Supplier:** slightly higher total risk. The primary risk is competitive pressure from rival SoC Suppliers that integrate against the same open-source middleware. Differentiation shifts from platform-specific software toward silicon performance.

4 | Conclusion

The strategic case for participation in open-source middleware rests on a differentiated verdict across four hypotheses. Lifecycle effort drops by up to 40% through joint development and the elimination of license fees. Combined development and adaptation time falls by up to 20%. Software OEMs cannot realize time effects because their in-house baselines are already optimized. Product-lifecycle stability is equivalent to proprietary middleware once the safety case is built. Total lifecycle risk is no greater than in-house development, but exposures shift from project-specific technical debt to shared governance dependence. Reducing the verdict to a 40% cost cut or a 20% time saving misses the structural shift: competition moves away from non-differentiating middleware toward the speed and quality of adaptation on top of a shared baseline, and the architectural influence to shape that baseline is a closing window.

Software OEM can participate but need to question themselves if the relative low cost savings of ~20% are worth the shift in risk profile and possible loss of competitiveness when giving up one of their core advantages. This verdict is supported by the rather passive attitude of most players within this archetype.

Legacy OEM in SDV Transformation are the main players for open-source development. They possess the capabilities needed and benefit enough from an open-source middleware available to all players. With time savings between 10–20% and cost savings doubling those numbers, the economic advantages outweigh risk shifts for most OEMs. The only question that remains is whether it is worthwhile to be the frontrunner and actively contribute, or whether even bigger results can be achieved through strategic positioning as a mere user of the finished middleware.

Technology Laggers dependency on Traditional System/ Middleware Supplier does not prevent them from realizing significant benefits as well. Significant savings in both time and cost can be achieved, provided they partner up with a supplier who can handle the adaptation and maintenance.

To fulfill this role, Traditional System/ Middleware Supplier are facing the biggest change. The traditional business models based on licensing fees for software and development services are being rendered obsolete by open-source middleware. A strategic positioning as an integrator, taking on adaptation and maintenance tasks as a long-term partner to multiple OEMs, could represent a new business model. However, this requires an upfront investment in the form of active participation as a contributor within the community and lead to, even with strong market positioning, a possible decline in revenue for this archetype if the open-source solution gains widespread adoption.

Hyperscaler can further strengthen their already comfortable position. The open-source middleware's standardized interfaces to the cloud make it even easier to scale the products across vehicles, architectures and OEMs. The higher-margin revenue streams from ongoing support and cloud usage remain unchanged. Thus, in most cases, the ideal strategy is to design the products to be compatible with the new interfaces.

The same holds true for most SoC supplier. Optimized scaling via fitting and supported hardware solutions is key. Building up an own ecosystem with related middleware and toolset is only a viable path, if the strategic positioning and capabilities of the own products is good enough to compete with a significant cheaper open-source solution. Only few SoC suppliers are in such a position and therefore following such a strategy.

Two patterns connect the six archetype recommendations. First, every recommendation is conditional. Open-source middleware rewards active participation. The cost saving, the time saving, and the risk equivalence all assume that the actor in question is contributing when the baseline architecture is decided. The 40% lifecycle effort saving evaporates for an OEM with a fundamentally different E/E architecture; the time saving for the Technology Lagger evaporates if the supplier was not contributing in the development; the Hyperscaler scaling benefit reduces if the cloud-protocol interfaces standardize in favor of its competitor.

Second, the transition compresses into a window measured in months, not years. First middleware will go live within the next 12 months from now. The marginal cost of joining late rises sharply. Integration debt against a fixed baseline, lost governance influence and the inability to retro-fit architectural preferences that should have been negotiated upfront are to be considered in the same way as a migration- and transition phase from current stacks to open-source based ones.

Standing aside is a defensible position only for actors that explicitly rank short-term capital discipline above long-term architectural influence. For OEMs and suppliers whose competitive position depends on the software stack they ship, that ranking inverts the strategic priority. The window to shape the baseline closes within the next months, after that, the architecture lives with whatever was decided in the room, for the rest of the product cycle ahead.

To discuss your role in the emerging open-source middleware ecosystem, contact your Berylls by AlixPartners team for strategic and operational guidance.

Authors

Dennis Röhr
Partner & Managing Director
Berylls by AlixPartners
dennis.roehr@berylls.com

Malte Broxtermann
Partner
Berylls by AlixPartners
malte.broxtermann@berylls.com

Andreas Maihöfner
Associate Partner
Berylls by AlixPartners
andreas.maihoefner@berylls.com

Rafael Wenger
Senior Consultant
Berylls by AlixPartners
rafael.wenger@berylls.com

Georg Weinberger
Senior Consultant
Berylls by AlixPartners
georg.weinberger@berylls.com

Berylls by AlixPartners
Questions? Contact us.

info@berylls.com
berylls.com

berylls by AlixPartners