

The background features a dark blue gradient with glowing, light blue circuit-like lines that branch out from the right side towards the left. Overlaid on the left side is a list of blurred, semi-transparent text entries, each appearing to be a set of coordinates or data points enclosed in curly braces, such as {111.6 527.8 4.3 -1.6 4.3}.

The software-defined vehicle is won or lost in the operating model

Design principles, concrete concepts and a transformation approach for OEMs and Tier X suppliers

October 2026



SUMMARY

The software-defined vehicle (SDV) fails not for lack of technology spend but because incumbents run a new product through a hardware-era R&D operating model. The way through is to read the design principles off the product itself, apply them to the four operating model elements that carry the change (Architecture & Technology, Capabilities, Governance including organization, and Way of working), and build the result clean sheet in a protected unit rather than retrofitting it onto the legacy architecture.

1

Software already decides which cars stay on sale



Technology disruption, shifting customer expectations, tightening regulation and geopolitical fragmentation are moving the industry from hardware-centric vehicles to software-defined vehicles.

The above mentioned shift changes what the product is, where value sits, and how cars are built. Regulation on its own can now decide whether a model stays on sale. Porsche took the combustion Macan off the European market because the car could not be brought up to the UN cybersecurity requirements (UNECE) that became mandatory for new registrations in July 2024. A software and architecture problem presented itself as a product decision.

Most SDV transformations stall for a different reason. Companies fund the new technology and then run it through the operating model built for hardware. The product changes but the way of working does not. It is one reason why almost every major OEM with a large-scale SDV program is now running behind schedule.

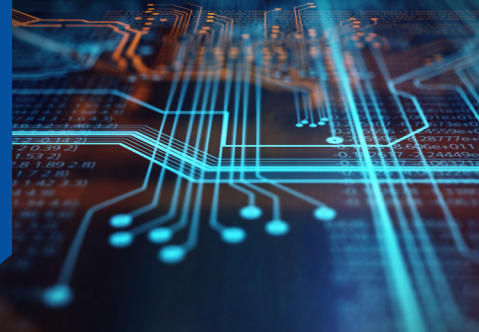
Tesla and BYD make the point from the same side. Both run flat structures with short decision paths, few layers between the engineer and the person who decides, and one software mainline that sets the pace for everyone working on the car. Xiaomi arrives at the same place from another direction, carrying a deeply integrated stack from silicon and operating system into the vehicle and reusing what it already ran in consumer electronics. None of the three copied a legacy template. Each built an organization that fits its product, and that fit, rather than a universal blueprint, is what the next era demands. Where product, processes, organization and partnerships are aligned, speed, scale and cost discipline reinforce one another.

This paper looks at the shift through an R&D lens. That is a deliberate premise rather than a claim that R&D carries the whole task. Procurement, financial management, quality and after-sales are affected with comparable force, and the two interfaces that most often decide whether the R&D changes hold are picked up later in the text.



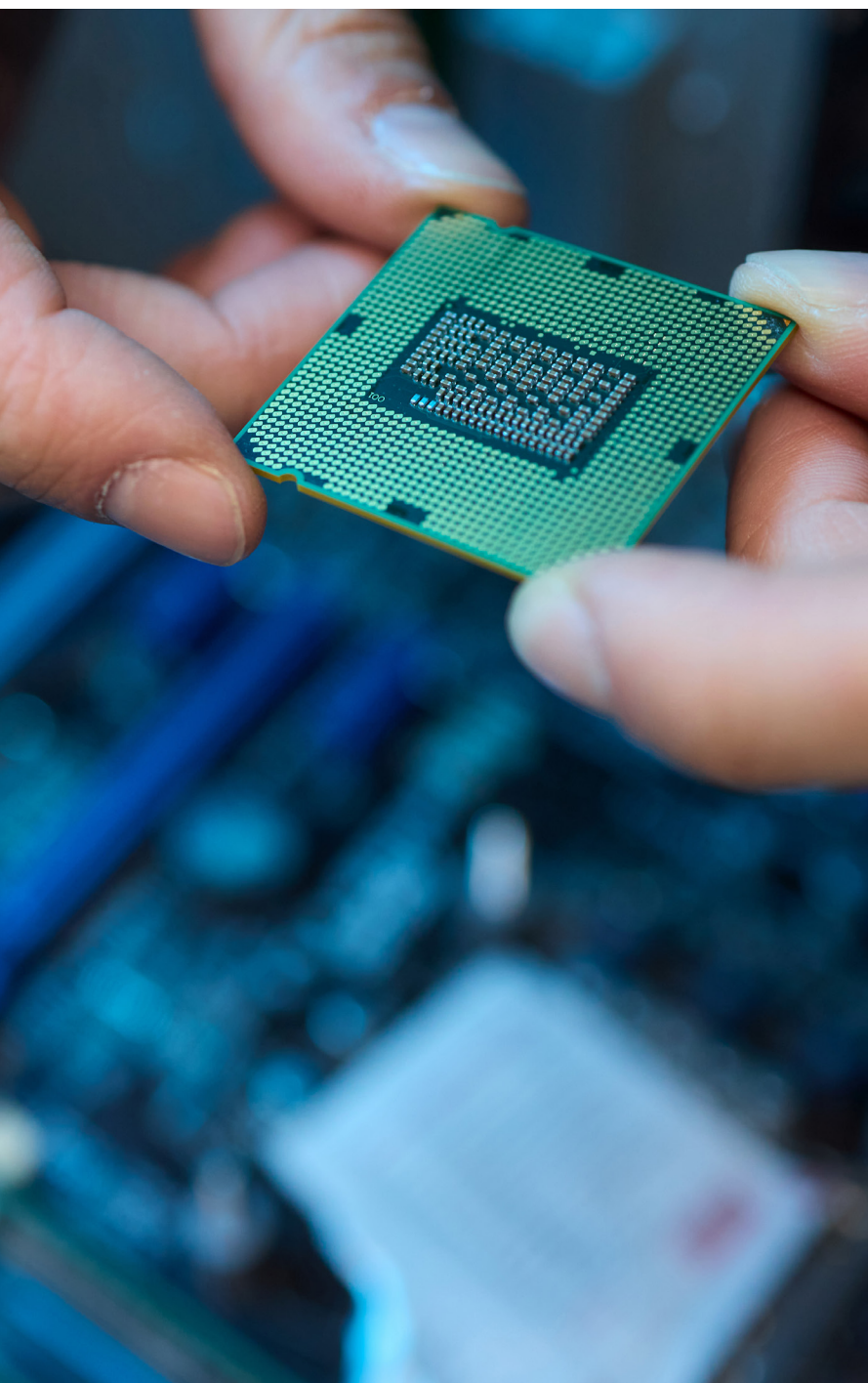
2

Value has moved into software, and the old operating model cannot carry it



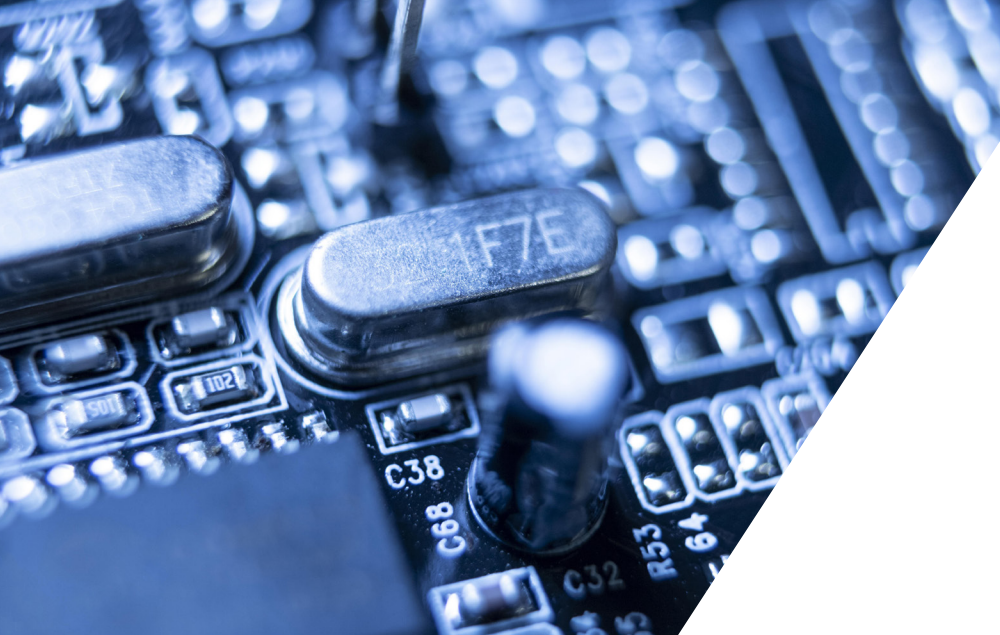
The operating model must change because the product already has. Historically, vehicle value was fixed at engineering freeze, with software embedded in and subordinate to hardware.

SDVs reverse that. Software carries the value and differentiates cars built on the same physical platform. Four shifts, in electronics, in software, in lifecycle, in artificial intelligence and the product itself, are what make legacy operating models incompatible with how these vehicles are built.



E/E ARCHITECTURE.

E/E architecture (electrical/electronic) is moving from many single-function, supplier-owned ECUs (Electronic Control Units) to zonal consolidation and centralized high-performance compute, which is what makes software-defined features possible. In the old distributed setup, a single over-the-air function could require coordination across more than 80 controllers and dozens of suppliers. BMW's Neue Klasse concentrates functions into four central "Superbrains" with more than 20 times the compute of today's cars, while a zonal harness removes about 600 meters of wiring and cuts harness weight by 30%. The number of control units is not what decides the outcome. Consolidation pays off only when it comes with one software mainline that every team integrates into on the same rhythm, because the mainline is what sets the development cadence. Without it, a company ends up with fewer boxes and the same fragmented way of working.



LIFECYCLE & CONNECTIVITY.

The product no longer freezes at SOP (Start of Production). Connectivity and over-the-air updates keep vehicles evolving through new features, diagnostics and fixes. Revenue shifts from a single sale to value earned across the ownership period, which means carmakers must manage software versions across a live fleet.



SOFTWARE ARCHITECTURE.

Traditional firmware ties each software change to a hardware cycle. SDVs decouple the two through a layered stack, so code can be deployed independently, reused across platforms, and run on a common runtime. Development consolidates into a single mainline rather than several branches maintained in parallel. Hardware becomes the commodity and software the differentiator.



AI AND DATA.

AI turns the product from a deterministic into a probabilistic machine. As machine-learned models are integrated into perception, driver assistance and in-cabin functions, behavior is no longer fully specified in code but learned from data, so the same input can produce different outputs as models are retrained. That undermines the classic assumption that a fixed test proves a fixed function, and it makes data pipelines, model training and continuous revalidation permanent parts of development rather than additions to a frozen software stack.

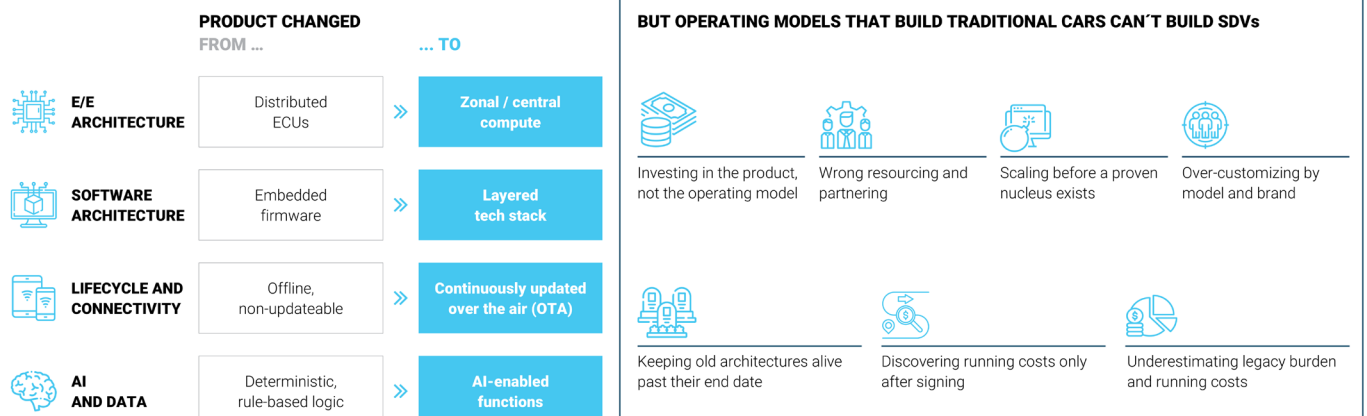


Exhibit 1: Product changes and the operating model failures that follow.

3 Programs fail in the operating model, not in technology



Spending on SDV programs has been substantial. In recent years, several major OEMs committed multi-billion budgets to new SDV platforms, only to postpone or cancel them, usually because of spiraling costs or delays in delivering parts of the platform. The same mistakes recur across the industry, and they rarely arrive one at a time.

One of the first mistakes is money going into the product but not into the operating model. Companies fund zonal architectures, central compute and AI stacks while processes, governance and incentives stay exactly where they were. Programs still run through the old stage gates, P&L stays tied to SOP, and agile teams report into waterfall hierarchies. New technology gets built in the old world.

Progress then comes in slower than promised, and the reflex is to scale anyway. Leadership rolls out semi-ready technology company-wide, freezes the legacy organization before the replacement works, and asks the same teams to carry old and new products at once. With no nucleus that has proven itself, mistakes scale faster than capability, and credibility goes with them.

The legacy burden shows its true size next. Retiring ECUs, toolchains and variants takes longer and costs more than planned, and retiring a whole architecture is harder still. Few organizations dare to switch an old architecture off on schedule, because a vehicle program leader will always have a reason to protect the launch of their own car, and granting that exception is easier than carrying the risk. The architecture stays alive well past the date it was meant to end, and the cost of running two worlds is carried quietly by engineering. Where backwards compatibility is missing, the installed fleet breaks. Where it is carried indefinitely, engineers stay locked in maintenance while new features starve.

Resourcing and partnerships add to the pile. Legacy engineers are asked to deliver software domains they have never worked in, hiring for the missing skills lags, and partners are managed as a procurement exercise rather than as joint engineering. The running costs of an SDV, from over-the-air infrastructure and cloud to cybersecurity, release management and support, tend to surface only after the commitments have been signed.

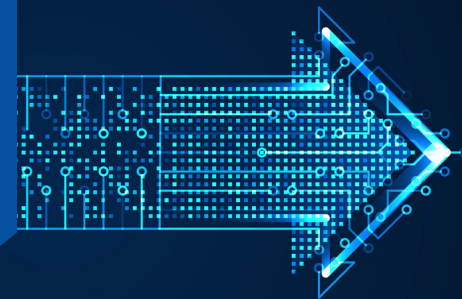
Over-customization closes the circle. Teams differentiate by model and by brand, rebuilding inside the platform the complexity the platform was meant to remove. Differentiation belongs in the software application layer.

None of these patterns yields to technology alone. Each of them is a property of how the company decides, organizes, measures and buys.



4

Eight design principles set direction a Software target operating model must take



Reading back from the product changes, eight design principles recur in the transformations that have worked. They describe the direction a Software target operating model must move in, and they set the criteria for redesigning every part of it.

OPERATING MODEL DESIGN PRINCIPLES

- 1 Vehicle line to **feature focused**
- 2 SOP to **continuous delivery**
- 3 Push to **pull principle**
- 4 Command-and-control to **autonomous teams**
- 5 General manager to **chief architect**
- 6 Resource ownership to **outcome accountability**
- 7 Black box to **source-code collaboration**
- 8 Manual processes to **everything-as-a-code**

SOFTWARE TARGET OPERATING MODEL

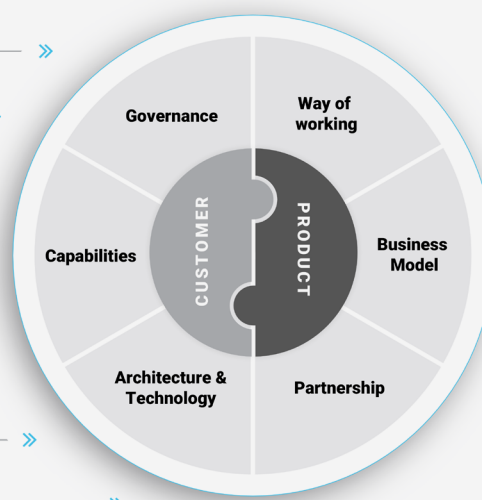


Exhibit 2: Design principles and six elements of the Software target operating model.

1 VEHICLE LINE TO FEATURE FOCUSED.

Development is organized around features and feature domains instead of around hardware modules and the vehicle line that owns them. A feature such as automated parking, or a driver profile that follows the customer from one car to the next, cuts across sensors, compute, body electronics and cloud services, so a team scoped to a single module can only ever deliver a fragment of it. The feature becomes the unit that gets planned, funded and staffed, prioritized by business value and customer demand rather than by the SOP of one vehicle project. Structure, funding and measurement all follow that unit.

2 SOP TO CONTINUOUS DELIVERY.

SOP stops being the finish line. From the day the first car is built, the company takes on a second obligation alongside the hardware launch: a substantial software stack that must be maintained, extended, validated and shipped for as long as the fleet is on the road, and across vehicle generations. That obligation needs permanent CI/CD and virtual validation capacity, planned, staffed and funded as a standing commitment rather than as the tail end of a project.

3 PUSH TO PULL PRINCIPLE.

Feature priorities come from voice-of-customer input and fleet data rather than from a fixed internal roadmap pushed out on a project cadence. The market becomes a continuous input instead of a periodic checkpoint.

4 COMMAND-AND-CONTROL TO AUTONOMOUS TEAMS.

Small, cross-functional teams own feature domains end to end, with clear mandates and short feedback loops. Decisions are taken at team level rather than routed up an approval chain.

5 GENERAL MANAGER TO CHIEF ARCHITECT.

Architectural judgment on platform design, technology choices and integration trade-offs belongs in leadership rather than delegated to specialists two levels down. That sets a hard requirement for who holds the job. Leaders need technical experience of their own, enough to

read a design review, challenge an interface decision on its merits, and step into the substance personally when a program is in difficulty. A manager who can only convene the experts cannot make the call, and in a feature-driven organization the call rarely waits. This is a talent decision and a governance decision at the same time.

6 RESOURCE OWNERSHIP TO OUTCOME ACCOUNTABILITY.

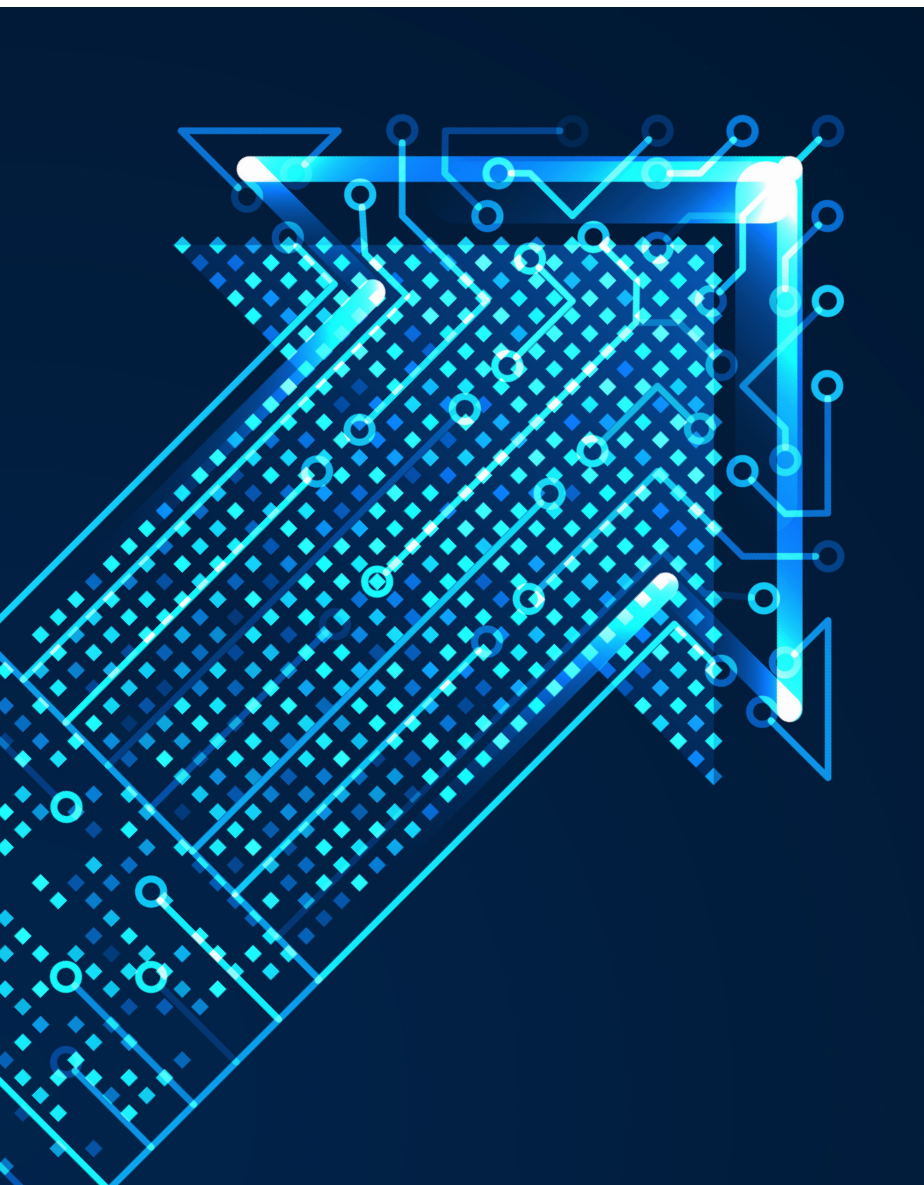
Headcount, budget and span of control are the wrong proxies for influence in an SDV world. Managers are held accountable for feature delivery, customer outcomes and platform quality instead.

7 BLACK BOX TO SOURCE-CODE COLLABORATION.

OEM and Tier X teams work together at source-code level, on shared toolchains and CI/CD pipelines, with IP (Intellectual property) and liability settled before the first line is written. This is the precondition for a single mainline. As long as suppliers hand over sealed binaries on their own release calendar, nobody can integrate the whole product daily, and the mainline stays a diagram rather than a practice. It is a structural change to sourcing, not a process tweak.

8 MANUAL PROCESSES TO EVERYTHING AS CODE.

Infrastructure, configuration, build and release logic, and the safety, cybersecurity and ASPICE (Automotive Software Process Improvement and Capability Determination) checks all live in version control and run in the pipeline. Every build stays traceable and can be rebuilt and re-verified years after SOP. Compliance becomes a property of the pipeline instead of a document produced alongside it.



5 Four elements carry the change, and they hold only in sequence



Principles set direction. The hard part is turning them into concrete choices about how decisions are made, how people are organized, and what the work runs on.

The Berylls by AlixPartners Software target operating model makes that change concrete, and four of its elements sit within R&D's own remit, which is where this paper focuses. Because the terms are used loosely in practice, it is worth fixing what each one covers here.



Exhibit 3: The six elements of the Software target operating model while this paper focuses on the four highlighted.

ARCHITECTURE & TECHNOLOGY. The platform and tooling base everything else runs on, including the repository, the pipeline and the validation environment.

CAPABILITIES. The skills, roles and career structures that make the other five workable, and the hiring and promotion decisions that signal what the company values.

GOVERNANCE. Who decides what, at which cadence, and along which escalation path. Boards, mandates, milestones, decision rights, and the interface between the hardware clock and the software clock sit here, together with the organizational structure that carries them.

WAY OF WORKING. How a single feature moves from requirement to release and beyond: the roles involved, the phases it passes through, the artefacts it leaves behind, and the handover once it is live.

BUSINESS MODEL. How the company earns from the product it now builds: what is priced, on what basis, and how revenue accrues across the ownership period rather than at the point of sale.

PARTNERSHIP. How work is divided and joined between OEM and Tier X, from the sourcing model to shared pipelines, joint teams, and the treatment of IP and liability.



This paper takes an R&D perspective and therefore concentrates on the four elements an R&D organization can redesign within its own remit: Architecture & Technology, Capabilities, Governance and Way of working. Business Model and Partnership matter just as much to the overall result, but they are shaped largely outside R&D, by sales, finance and procurement, and are addressed here only where they touch the four. Crucially, these four do not change in isolation, they reinforce one another in a clear sequence. Architecture & Technology foundations such as everything as code, a single repository and continuous delivery, together with capabilities such as the chief architect role and trained skill pools, make a governance model possible that steers by feature and architecture rather than by individual vehicle programs. That governance is what allows a different Way of working, with E2E delivery in autonomous, dynamically formed feature teams that own a feature from requirement to release.

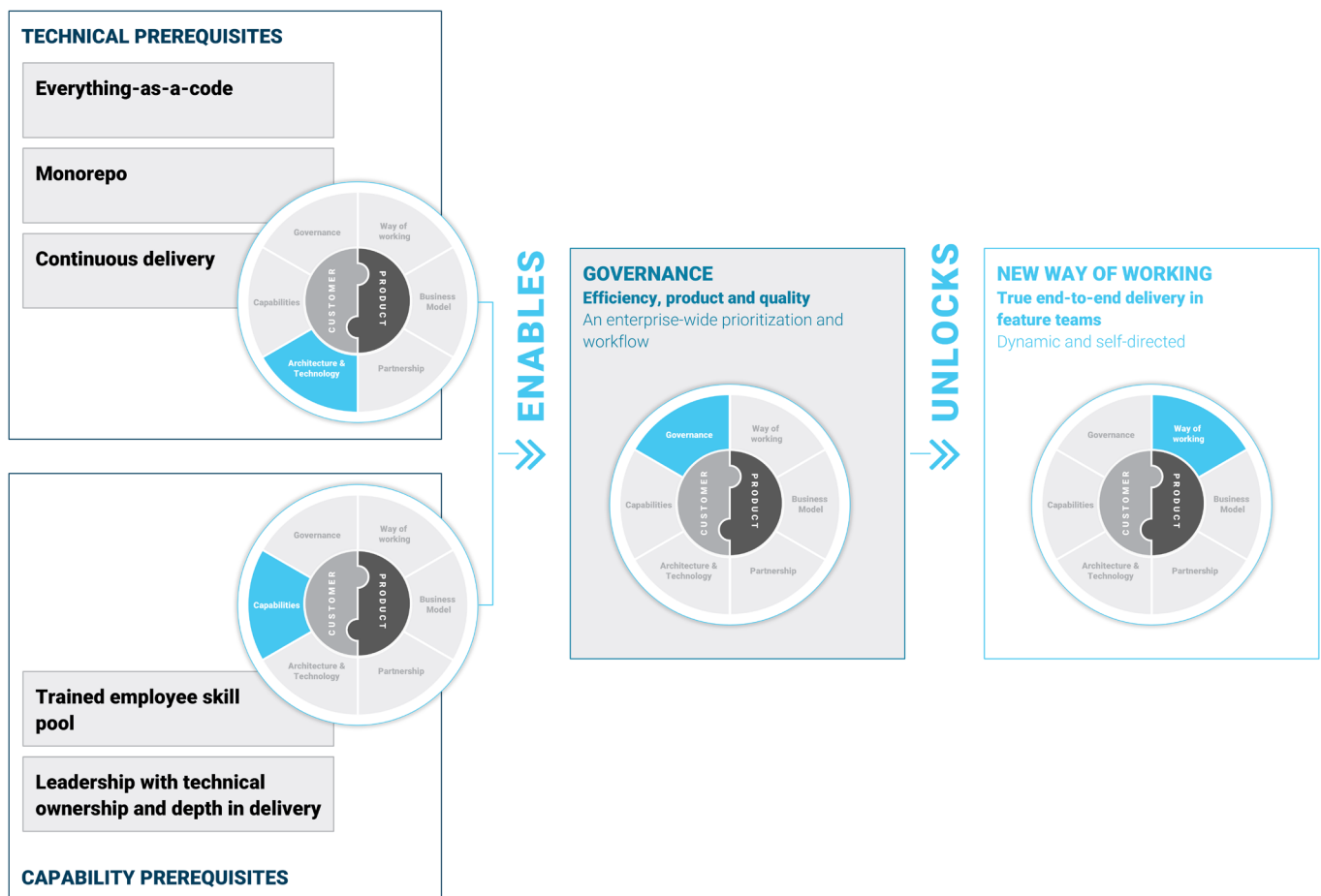


Exhibit 4: Technical and capability prerequisites enable the governance that unlocks the new Way of working.

5.1 Architecture & Technology – The tooling base decides whether the principles survive contact with delivery

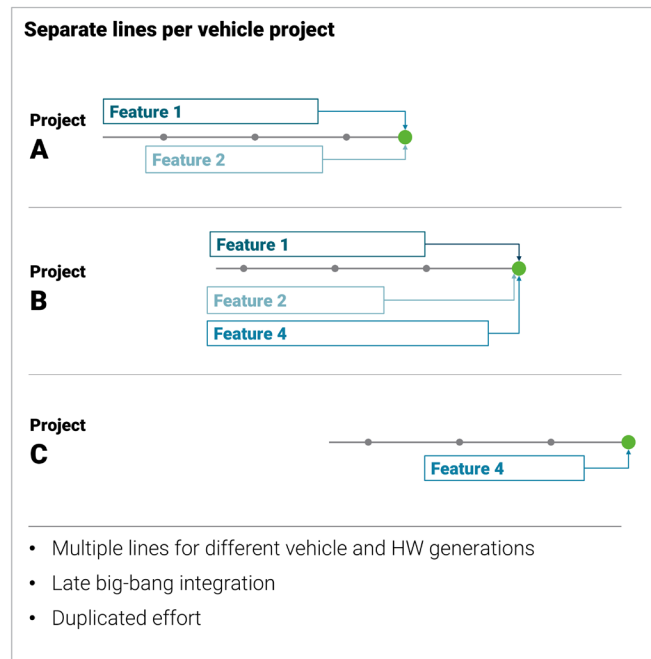
This is where the principles either hold or fall apart. Continuous delivery and source-code collaboration do not survive on good intentions. They need a tooling base that makes the right way the path of least resistance. Two choices do most of the work.

Everything as code comes first. Infrastructure, configuration, build and release logic, and the safety, cybersecurity and ASPICE checks all live in version control and run in the pipeline, so any build can be traced, rebuilt and re-verified years after SOP.

A single repository comes second. Pulling vehicle software into one mainline as the single source of truth is what makes the daily merge meaningful, exposes cross-team and supplier dependencies in the code rather than behind contracts, and lets a feature built once serve the whole portfolio.

Both choices need someone senior enough to defend them when a program wants an exception.

OLD



NEW

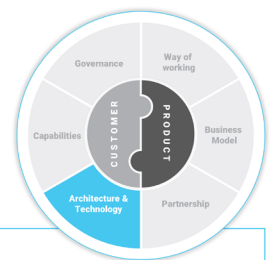
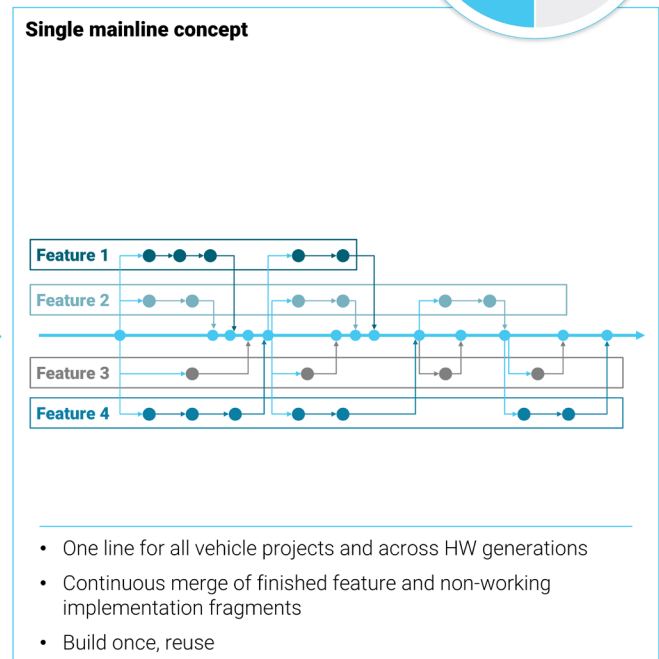


Exhibit 5: A mainline per vehicle project against one shared mainline integrated continuously.





5.2 Capabilities – Promotions are the clearest statement a company makes about what it values

Technical depth replaces program-management seniority as the main qualification for senior leadership. The chief architect reports to the CTO, sits at C-1 level, and is paid like it. A genuine expert career track runs alongside the management track, with comparable pay, status and decision rights.

OLD

General Manager

Architecture & System Thinking	
Cross-Domain Expertise	
Soft skills	
Core technology expertise	
New economy experience	
Commercial & P&L acumen	
Zero-defect mindset	

- Program steering
- Vehicle project optimization
- No hands-on role in product development

NEW

Chief Architect

Architecture & System Thinking	
Cross-Domain Expertise	
Soft skills	
Core technology expertise	
New economy experience	
Commercial & P&L acumen	
Zero-defect mindset	

- System and technology expertise
- System optimization
- Hands-on technical authority



Exhibit 6: Competency profile of the general manager against the chief architect.

Hiring shifts toward software and platform backgrounds, including from hyperscalers and tech-native players, rather than from traditional program leadership. None of this is symbolic. Promotions are the clearest signal a company sends about what it values, and if the top jobs go to generalists, engineers will read the message and behave accordingly.

5.3 Governance – Three layers keep the hardware and software clocks in step

A feature-based organization that delivers continuously against a decoupled architecture cannot run on the program-bound, hierarchical decision-making of the hardware era. The central tension is speed. Hardware moves on long, physically constrained milestones, software on short continuous cycles. The SDV governance model runs both clocks deliberately, across three layers, each with its own cadence, its own owner, and a defined interface to the layer above.

LAYER 1: Platform architecture. A Platform Architecture Board chaired by the chief architect owns the cross-domain choices: compute topology, operating system and middleware, the communication backbone, and the hardware abstraction layer. These follow platform requirements rather than supplier timelines or the cost target of a single program, and they are set on a multi-year horizon so that several programs and feature generations can run on one architecture version. This sets the long-horizon frame both clocks run within. The decisions are taken once per architecture generation and then held stable, which is precisely what gives the layers above something to build on.

LAYER 2: Program-platform interface. Each vehicle program commits at defined milestones to one platform version as its build target. The platform organization keeps that version stable for the life of the program while the mainline keeps moving, in effect a long-term-support branch. Programs adopt platform releases, they do not co-develop them. This is where the hardware clock ticks on vehicle milestones, and the commitment is made once per program rather than renegotiated continuously.

LAYER 3: Feature coordination. This is the software clock, and it runs for as long as the software is in the field. Every feature cycle opens with an architectural and hardware-constraint review before design starts. Features that require hardware changes trigger a formal impact assessment, and features that span domains go to E2E teams. Hardware constraints become inputs to design rather than **surprises discovered at integration**.

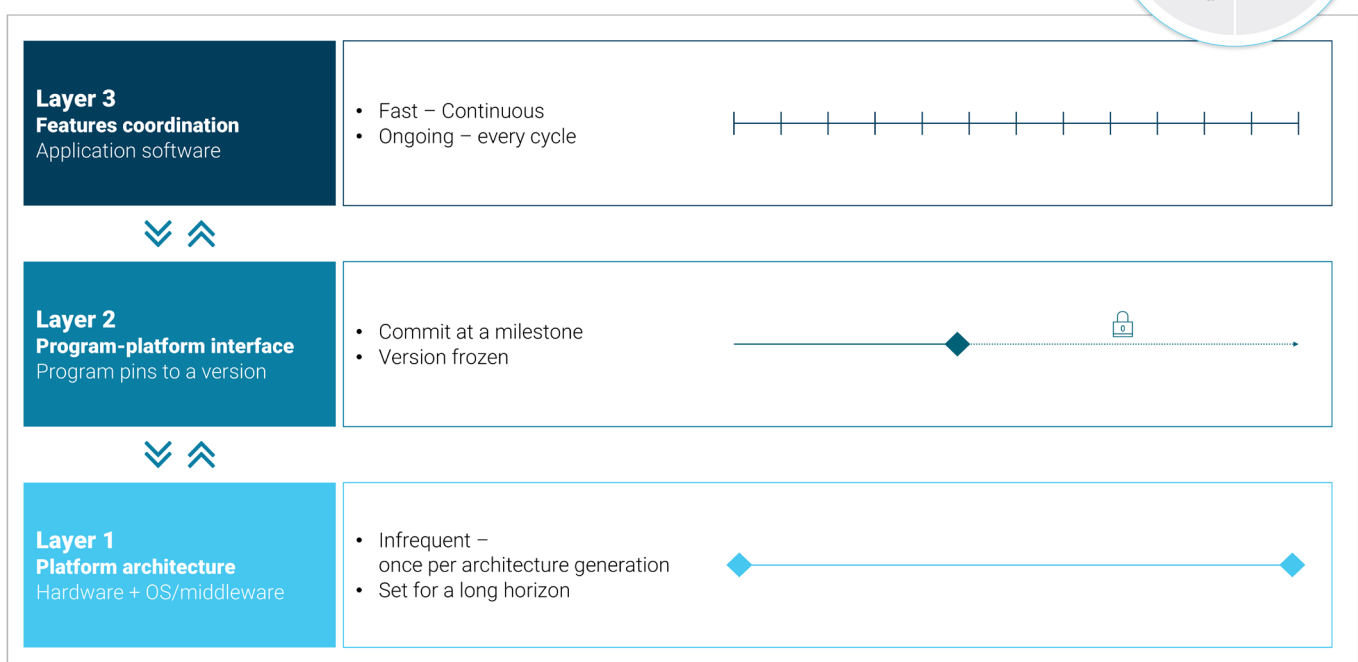
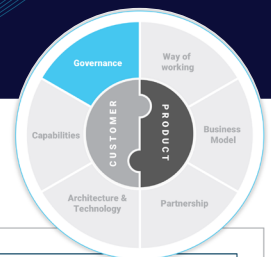


Exhibit 7: The three cadences, a long-horizon platform commitment, a version pinned per program, and continuous feature delivery on top of both.

Each layer works within the one above, and anything with architectural implications escalates rather than being absorbed as hidden technical debt. Three mechanisms hold it together: a decision register that records what was decided and why, a protocol that gets hardware changes to affected teams within a set time, and a cross-domain retrospective at each milestone.

Governance sets how decisions are made across platform, program and feature. It does not work on its own, though. Each layer needs a part of the organization that owns it and is set up to carry it out. Most OEMs still run on the component-based line organization of the hardware era. There, platform decisions are negotiated between component departments, and cross-domain features stall between them. This is why organization belongs inside governance, not next to it.

ORGANIZATION:

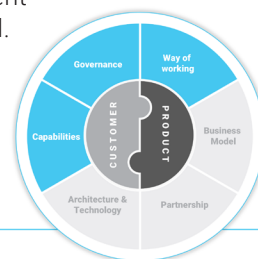
The component-centric line organization that built the combustion car stands directly in the way of feature-centric delivery. Matrix overlays, SAFe trains included, do not fix this, they just layer governance cost onto the same silos. A working SDV organization rests on four pillars and shifts the dominant team scope from one component, all features to one feature, all components. The result is a network with clear interfaces, not a matrix with escalation chains.

PILLAR 1:

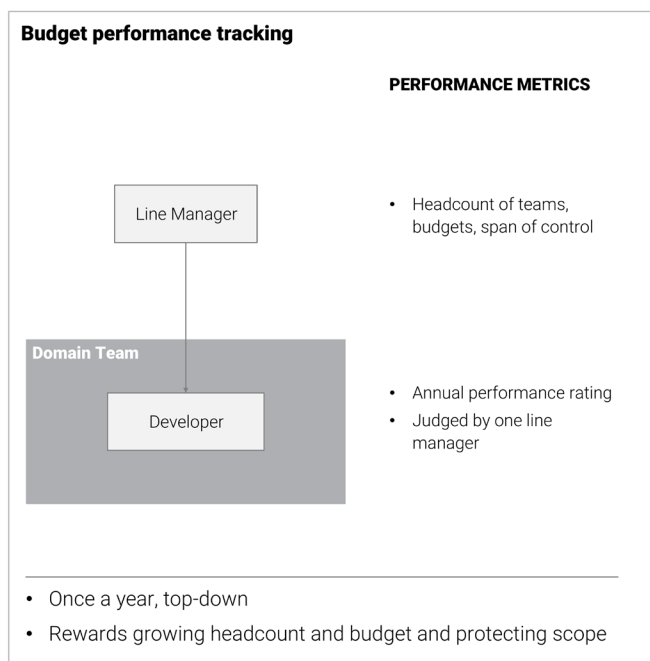
ARCHITECTURE FUNCTION. A central function owns the reference architecture, the interfaces, the technology stack and the architectural runway. What matters is its independence: its own budget, decoupled from any single program, so it can fund the runway ahead of need instead of reacting to whichever program shouts loudest. It is also the merge gate to a single repository (monorepo) holding one mainline where the product is integrated, tested and released.

PILLAR 2:

SKILL POOLS. Every developer belongs to exactly one pool, organized by capability: perception, cloud, middleware, V&V, requirements and so on. The pool owns hiring, skills and people management, but not delivery scope or P&L. That separation breaks the empire-building reflex that kills most matrix transformations, where managers grow a team to protect headcount regardless of need. Developers are staffed from the pool onto features and return to it once the feature is done. How often an engineer is pulled, and by which feature managers, becomes visible information: strong people are requested repeatedly, while those who are rarely requested surface a development or fit question that a classic line organization tends to absorb quietly. Utilization in feature development therefore serves as a performance measure for developers and as a capacity signal for the pool.



OLD



NEW

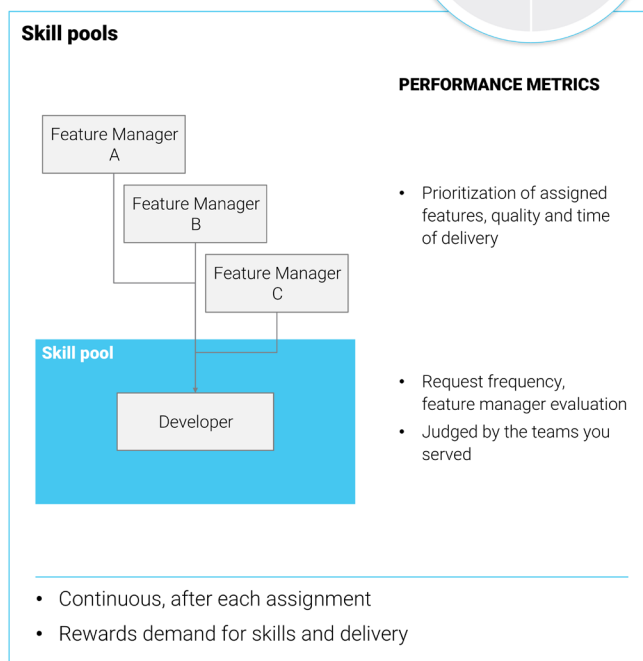
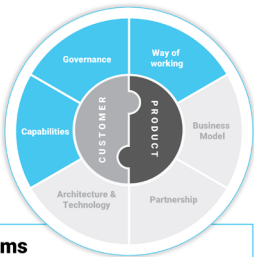


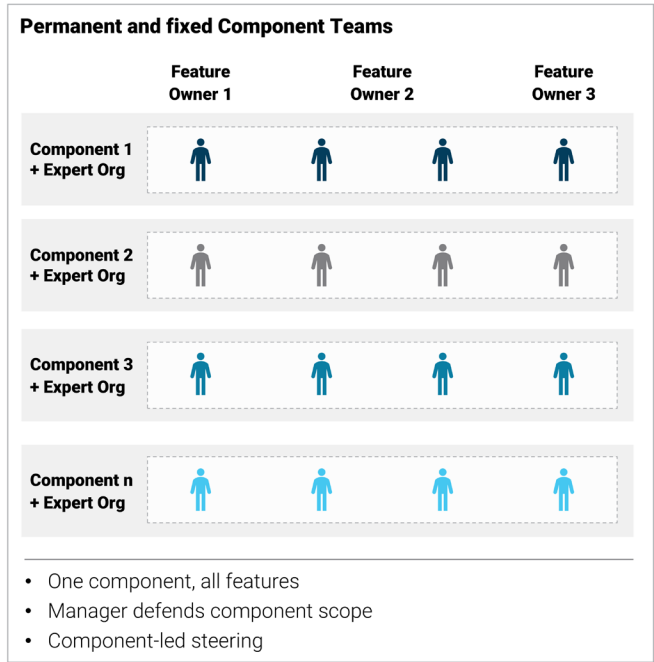
Exhibit 8: Line-managed domain teams against skill pools: how performance is measured and what each model rewards.

PILLAR 3:

E2E FEATURE TEAMS. Each prioritized feature gets a dedicated team drawn from every affected pool, carrying everything needed for delivery, from requirements, design, development, V&V and integration to release. It is time-bound. It forms when the feature is prioritized, delivers against the mainline, and dissolves once the feature is shipped and stable. An architect runs technical delivery while a feature manager carries accountability upward.



OLD



NEW

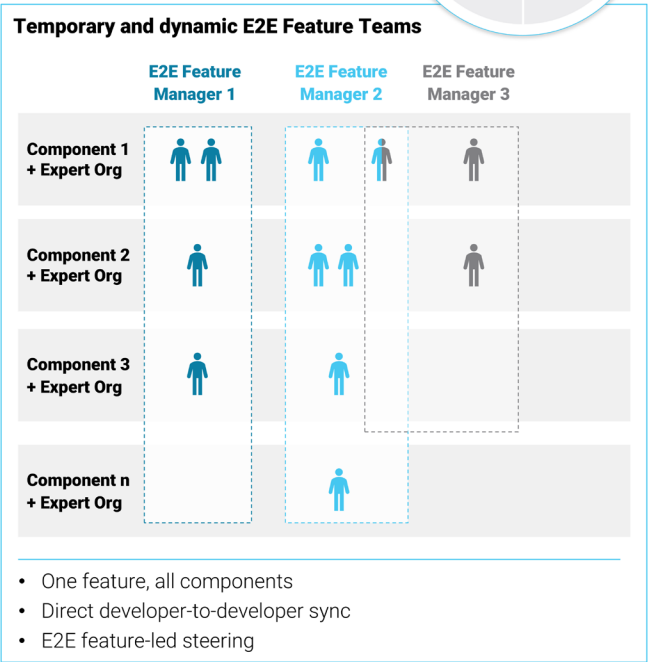


Exhibit 9: Permanent component teams against temporary E2E feature teams.

PILLAR 4:

ENTERPRISE PRIORITIZATION. Leadership keeps one unambiguous priority list, with no parallel lanes, no per-component quotas and no shadow backlogs. Feature managers pull developers from the pools in priority order. Anyone not pulled works on tooling or research in their pool, never on legacy programs. Reviewing the mainline, the priority list and the integration state becomes a CTO task, not a delegated PMO one.

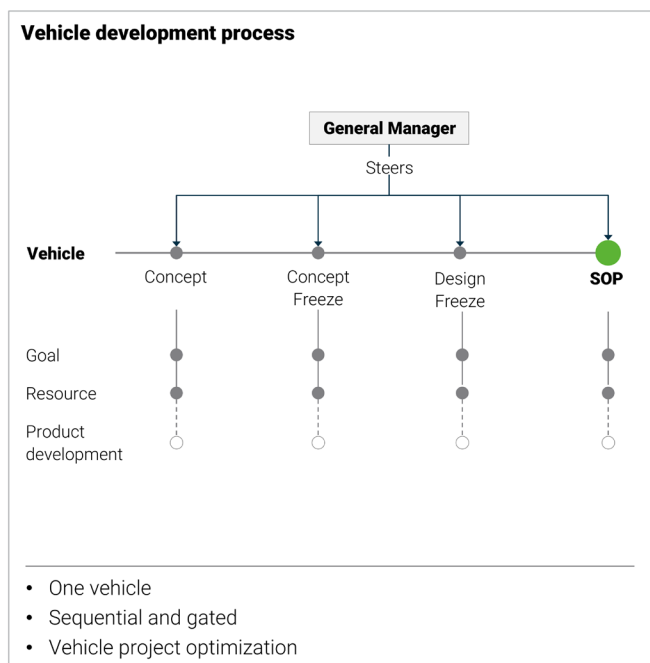
Together this produces an org chart with two stable layers, the architecture function and the skill pools, and a constantly reshuffling layer of feature teams on top, steered next to the CTO. People live in a pool, deliver in a team, the architecture function holds the system together, and enterprise prioritization decides the order of play.



5.4 Way of working – A shipped feature is not a closed project

Feature development, the third governance layer, is new ground for most incumbents. It runs on five roles, the enterprise leadership, the chief architect, the E2E feature manager, the developers and the skill pool managers, and it moves through five phases.

OLD



NEW

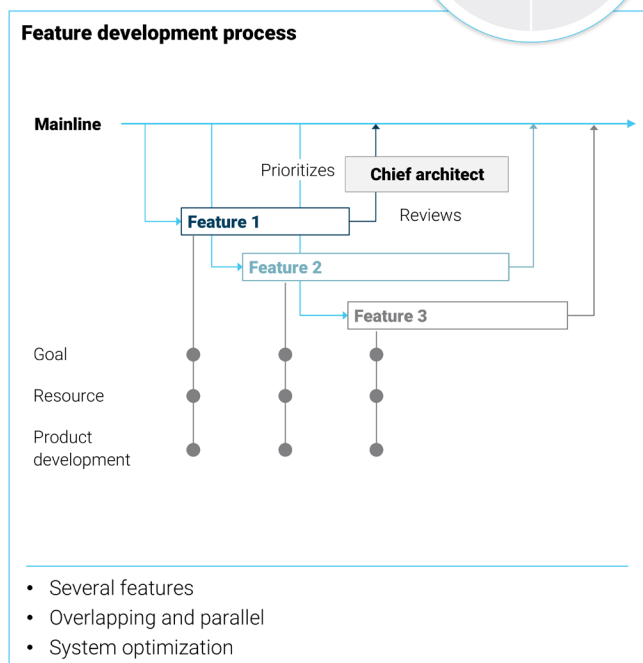


Exhibit 10: The gated vehicle development process against feature development on the mainline.

PLAN. Define product and timeline goals, break them into E2E feature increments, and put them in unambiguous priority order, checked against architectural, hardware and resource constraints.

ASSEMBLE. An architectural analysis decides which components and interfaces a feature touches, and the team is drawn from the relevant pools accordingly. Architecture precedes team formation.

BUILD. An optional design-review gate applies to the more complex features. Execution runs on fast cycles, quick impediment resolution, incremental delivery and a daily merge to the mainline, with all V-model artefacts kept on a monorepo feature branch.

MERGE. Before release, completed work passes a layered approval sequence, so architectural integrity, feature completeness and code quality are each signed off by the right authority before integration.

HAND OVER AND RECONCILE. The team does not simply disband. Ownership moves to a named home in the line organization, normally the skill pool or feature domain closest to it, and what transfers is the running software together with the architecture decisions behind it, the open defects, the test suites, and the cloud and operating cost the feature carries. Members then return to their pools, and resource use is reconciled against pool budgets to rebalance capacity for the next cycle.

Follow-up development is planned on the same basis. A feature in the field generates fleet data, defects, regulatory obligations and requests for extension, so the enterprise priority list carries maintenance and follow-up increments next to new features instead of treating them as overhead to be absorbed somewhere. When an increment is large enough, a team forms around it again out of the pools, starting from the architecture and test artefacts the previous team left behind. This is where the single repository earns its place: the code and the decisions recorded with it are the handover, which only works if the artefacts live in the mainline rather than with a team that has since dissolved.

Two changes decide whether any of this holds. First, measurement must follow the work, not resource ownership. Feature managers are judged on customer impact and adoption, pool managers on how often their engineers are pulled and the feedback those teams give, architects on the runway they build and the technical debt they remove. Without that realignment, incentives drag the structure back into a matrix within a single planning cycle. Second, the engineering-freeze mindset must give way to a software mindset. Release at sufficient maturity and iterate over the air, treat controlled failure as a source of learning rather than a career risk, and work with partners at source-code level. This is the hardest part to mandate, and the clearest line between Tesla and BYD and most incumbents.



6

Clean sheet in a protected unit, not a retrofit



No incumbent automaker has successfully become software-defined by upgrading its legacy architecture in place.

The most advanced examples come from new entrants like Tesla, BYD, Xiaomi and Rivian. Even incumbents recognize this: Volkswagen chose to invest up to USD 5.8 billion in a Rivian-based platform rather than continue its own system. The lesson is not that transformation is impossible, but that it must be built on a clean-sheet platform in a protected unit rather than retrofitted onto legacy systems.

Six moves follow as a guideline for implementing a new Software target operating model.



ARCHITECTURE FIRST.

Commit early to a centralized, zonal platform with a single software backbone, and design the organization around it. The gains are fewer ECUs, less wiring and faster development. If needed, partner for the platform, but retain architectural control.

START WITH A FOCUSED NUCLEUS.

Build around one vehicle with a clear launch date, not a broad multi-brand platform. Large-scale, “big bang” approaches consistently fail.

RING-FENCE BUT ENFORCE RESULTS.

Protect the unit with dedicated funding and authority and tie continued support to delivering concrete milestones.

TREAT DUAL MODELS AS TEMPORARY.

Running legacy and new systems in parallel is transitional. Define a clear trigger for migration to the new platform and end the split.

BLEND TALENT AND RETAIN IT.

Combine automotive and software expertise, give architects strong career paths, and protect culture from legacy metrics. Retention is the main challenge.

BASE THE BUSINESS CASE ON COST AND SPEED.

Focus on tangible gains like simplification and faster development. Treat subscription revenue as upside, not the core justification.

Transformation remains difficult and failure rates are high, especially when new products are built on old systems. These moves apply mainly to companies early in their transition. Companies that are already further along may no longer need a ring-fenced unit and should instead scale the new model across the organization.

CONTACT THE AUTHORS:



Christian Kaiser

Partner & Managing Director
Berylls by AlixPartners
christian.kaiser@berylls.com



Sebastian Bräuer

Partner
Berylls by AlixPartners
sebastian.braeuer@berylls.com



Mirko Mählich

Expert Advisor
Universität der Bundeswehr
mirko.maehlich@me.com



Tom Volber

Project Manager
Berylls by AlixPartners
tom.volber@berylls.com



Yonnas Tesfamicael

Senior Consultant
Berylls by AlixPartners
yonnas.tesfamicael@berylls.com



Mai Khoa Le

Consultant
Berylls by AlixPartners
mai-khoa.le@berylls.com

ABOUT US

For more than forty years, AlixPartners has helped businesses around the world respond quickly and decisively to their most critical challenges—circumstances as diverse as urgent performance improvement, accelerated transformation, complex restructuring and risk mitigation.

These are the moments when everything is on the line—a sudden shift in the market, an unexpected performance decline, a time-sensitive deal, a fork-in-the-road decision. But it's not what we do that makes a difference, it's how we do it.

Tackling situations when time is of the essence is part of our DNA—so we adopt an action-oriented approach at all times. We work in small, highly qualified teams with specific industry and functional expertise, and we operate at pace, moving quickly from analysis to implementation. We stand shoulder to shoulder with our clients until the job is done, and only measure our success in terms of the results we deliver.

Our approach enables us to help our clients confront and overcome truly future-defining challenges. We partner with you to make the right decisions and take the right actions. And we are right by your side. When it really matters.

The opinions expressed are those of the author(s) and do not necessarily reflect the views of AlixPartners, LLP, its affiliates, or any of its or their respective professionals or clients. This article "The software-defined vehicle is won or lost in the operating model" was prepared by AlixPartners, LLP ("AlixPartners") for general information and distribution on a strictly confidential and non-reliance basis. No one in possession of this Article may rely on any portion of this Article. This Article may be based, in whole or in part, on projections or forecasts of future events. A forecast, by its nature, is speculative and includes estimates and assumptions which may prove to be wrong. Actual results may, and frequently do, differ from those projected or forecast. The information in this Article reflects conditions and our views as of this date, all of which are subject to change. We undertake no obligation to update or provide any revisions to the Article. This Article is the property of AlixPartners, and neither the Article nor any of its contents may be copied, used, or distributed to any third party without the prior written consent of AlixPartners.