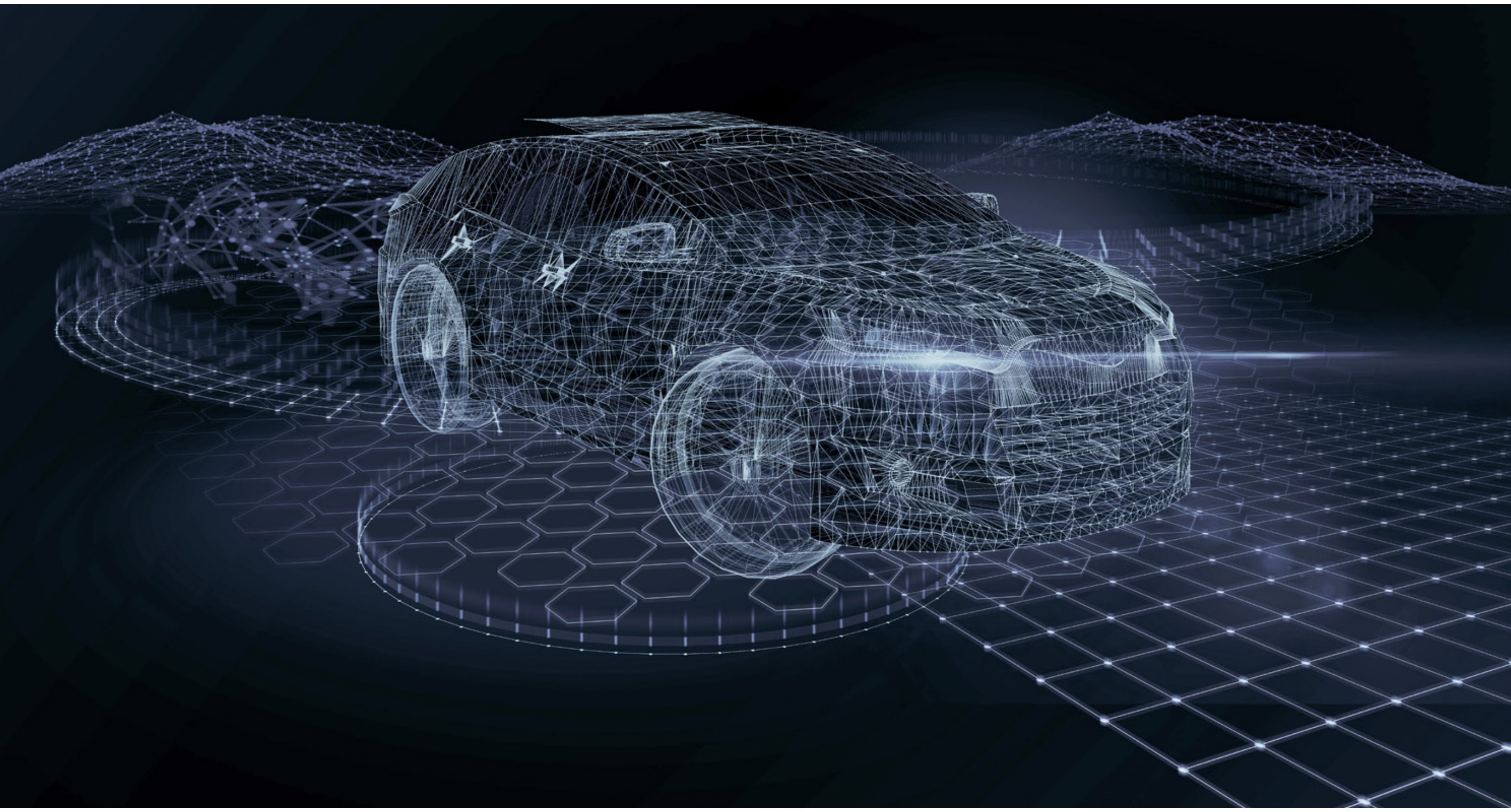
McKinsey & Company

McKinsey Center for Future Mobility

When code is king: Mastering automotive software excellence

With software driving tomorrow's automotive innovations, R&D organizations must quickly master its intricacies.

by Ondrej Burkacky, Johannes Deichmann, Stefan Frank, Dominik Hepp, and André Rocha



January 2021

Software is rapidly reprogramming the car industry. The four biggest disruptions in recent years—autonomous vehicles, connectivity, electrification, and shared mobility (ACES)—all rely heavily on leading-edge software. Further disruption in these areas lies ahead. OEMs, suppliers, and new players across the industry hope to capture critical control points in this new, software-driven value chain.

Software: A critical industry tipping point

As the landscape shifts, automakers that lack sufficient software capabilities will face major risks, including start-of-production (SOP) delays and budget overruns. They may also fall further behind competitors and new entrants that can bring far more innovative products much faster to the market. Even more troubling, software issues could lead to massive recalls or leave companies vulnerable to customer-safety risks resulting from hacking attacks.

Our research shows that the divide between strong software organizations and less capable groups is significant, with top companies reporting throughput and quality that is three to six times higher than that of bottom performers.[1] That is a far bigger productivity difference than the one between hardware developers. Many automotive players are aware of the advantages that come with strong software development and are now taking drastic steps to improve performance. Some are planning to step up their software capabilities and hire thousands of software engineers over the next few years, while others are redefining their governance models, setting up partnerships, and globally ramping up centers of excellence.

We believe these steps are insufficient, however, since real change will only come when automakers update their underlying operating models for software development. Based on our research, only 40 percent of the R&D leaders who view software as a major disruptor feel prepared to make the necessary operational shifts.[2] While leaders across industries have made step-change improvements in their software engineering practices, most automotive players still significantly lag behind high performers. Areas of concern include agile practices, continuous integration, and automated testing.

With so much at stake, automotive players should rethink their entire approach to software development, including the underlying operational model. This article shares our key beliefs and insights gained from working closely with automakers, suppliers, and other ecosystem partners. The insights are also based on information gathered through extensive

Based on our research, only 40 percent of the R&D leaders who view software as a major disruptor feel prepared to make the necessary operational shifts.

[1] Based on information from McKinsey's proprietary SoftCoster database, which includes more than 14,000 software-development projects across industries.
[2] McKinsey R&D of the Future Survey.

interviews with technology experts and a large-scale software benchmarking exercise involving McKinsey's proprietary SoftCoster database.

By the numbers: How software is taking over

Several trends highlight the growing importance of automotive software. The first involves the rapid expansion of the software and electrical/electronics market, which is expected to have a CAGR of 12 percent from 2020 to 2030—more than three times the expected growth for general automotive sales. Areas of strongest growth include software functions (with a CAGR of 11 percent) as well as integration testing (at 12 percent).[3]

Growing complexity, but slow productivity gains

The complexity of automotive software is escalating on both the functional and architectural levels, but development productivity is not rising at the same rate. Our research shows that software complexity grew by a factor of 4.0 over the past ten years, while software-development productivity increased by only a factor of 1.0 to 1.5 (Exhibit 1). The problem is most severe with large modules that are becoming increasingly complex, such as infotainment and advanced driver-assistance systems (ADAS). Productivity for these modules is about 25 to 35 percent lower than that of traditional, deeply embedded software.

This widening gap could compromise the future success of automotive players. Increased efforts for software development and maintenance over lifecycles may diminish their abilities to innovate and react to competitors. During our interviews, one company leader noted that software maintenance alone will rapidly use up all software R&D resources if complexity continues to grow while productivity remains unchanged, leaving little room for innovation. Ultimately, the complexity-productivity gap will reduce cost competitiveness and could lead to severe financial and reputational problems.

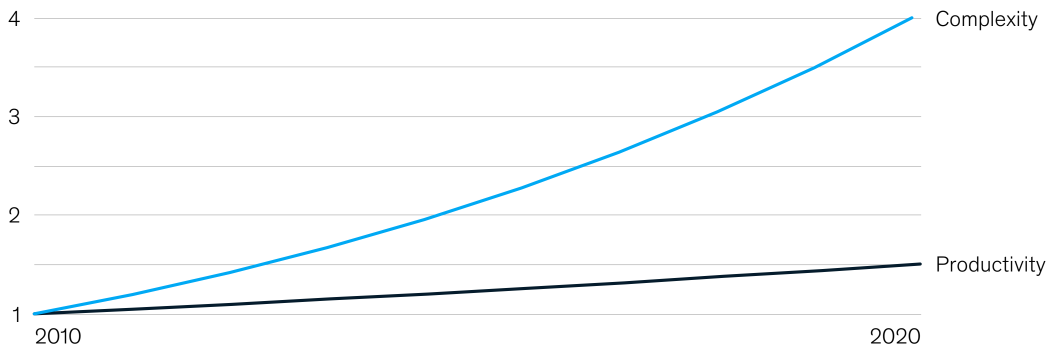
Significantly, organizations in the top quartile for software development achieve 3.0 times greater productivity compared with bottom-tier players, 3.5 times more throughput, and 6.0 times better quality (Exhibit 2). Consequently, their time to market is shorter and development costs are lower for each level of new software functionality.

[3] Ondrej Burkacky, Johannes Deichmann, Jan Paul Stein, *Automotive software and electronics 2030: Mapping the sector's future landscape*, July 9, 2019, McKinsey.com; updated publication forthcoming.

Exhibit 1

Software complexity is increasing more quickly than productivity.

Relative growth of software complexity and productivity over time, indexed for automotive features

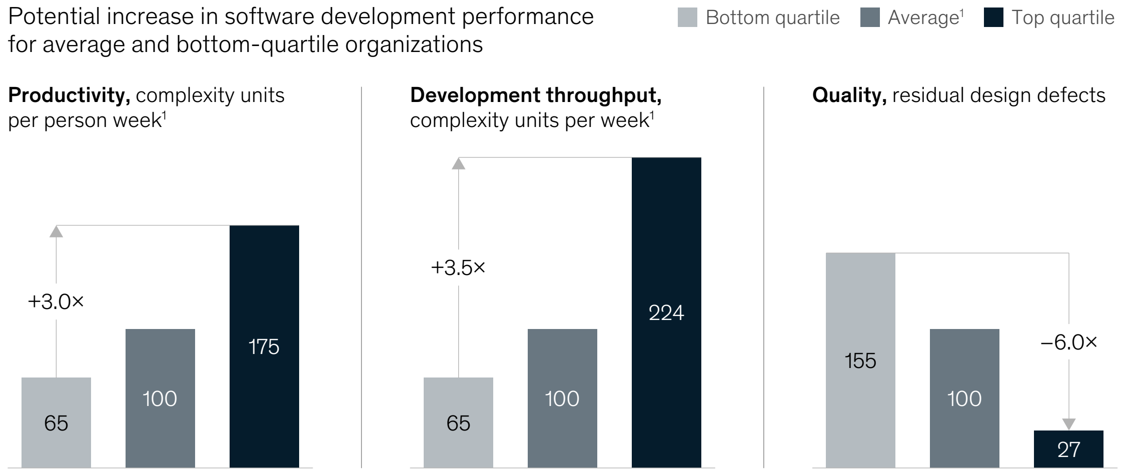


Source: McKinsey's SoftCoster embedded software project database

Exhibit 2

Companies in the top quartile for software development demonstrate significantly higher productivity, development throughput, and quality.

Potential increase in software development performance for average and bottom-quartile organizations



[1]Average indexed to 100. Complexity unit per person week is productivity per full-time employee. Complexity unit per week is productivity for the whole organization.
Source: McKinsey's SoftCoster embedded software project database

Performance differences between top and bottom companies are less pronounced with hardware, so there is less opportunity for differentiation in that area.

Slashing complexity while boosting efficiency

To succeed in this rapidly changing environment, companies should minimize complexity by reducing the effort required to develop and maintain software. This strategy will involve limiting how many versions of functions and features are available across platforms and life-cycle stages. Simulta-neously, companies must increase the reuse of components. For productivity, organizations should attempt to increase efficiency by matching the software-development speed of digital-native companies. Since the overall level of software innovation will not decline, companies must also increase their software-development and maintenance output to deliver the offerings currently required to be successful in the market.

Cutting complexity and increasing efficiency will require a new software operating model that focuses on four critical dimensions:

— *What software is developed,* including the architecture, design, and requirements

— *Where software is developed* within the organization, including locations, talent, and partnerships involved

— *How software is developed,* which considers development methodologies, such as agile-at-scale, or changes in development and testing processes

— *How software development is enabled,* including performance-management and toolchain infrastructure

The first dimension—what software is developed—focuses on reducing complexity through modular and decoupled hardware/software architecture, user-centered design, and requirements management. The other three dimensions focus on increasing the efficiency of software development by providing the right structures, processes, and infrastructure. We have identified 11 best practices across the four dimensions that can help automotive players successfully master their software challenges (Exhibit 3). Ideally, companies will address all dimensions simultaneously.

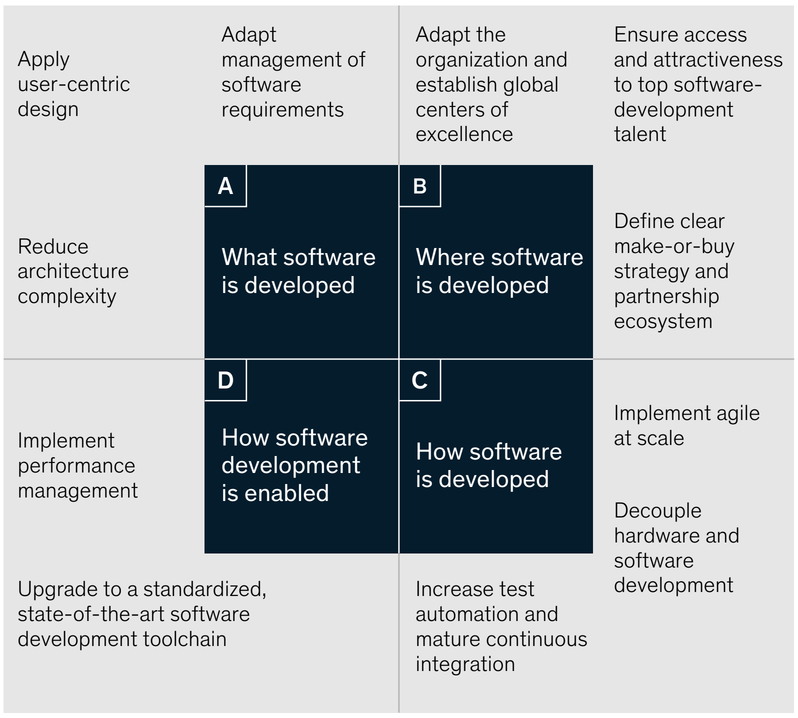
A. What software is developed: Architecture, design, and requirements

Under the new operating model, companies must translate their software-related aspirations and business opportunities into actionable architecture, product, and portfolio requirements at the product, function, and module levels. Through this process, companies gain a detailed understanding of the kinds of software that can create value for them. It also allows them to reduce architecture complexity, apply user-centered design techniques, and improve management of software requirements.

Exhibit 3

A new software operating model will require changes in four critical dimensions.

Best practices across dimensions


Apply user-centric design
Adapt management of software requirements
Adapt the organization and establish global centers of excellence
Ensure access and attractiveness to top software-development talent
Reduce architecture complexity
A
What software is developed
B
Where software is developed
Define clear make-or-buy strategy and partnership ecosystem
Implement performance management
D
How software development is enabled
C
How software is developed
Implement agile at scale
Decouple hardware and software development
Upgrade to a standardized, state-of-the-art software development toolchain
Increase test automation and mature continuous integration

A1. Reduce architecture complexity
Based on our research, a lack of modularity within automotive software drives higher design complexity which, in turn, increases overall project effort. What's more, automotive software often has suboptimal architectural component boundaries, which can lead to increased interdependencies that multiply the number of components developers must modify when adding new functionalities. These interdependencies also increase the time and expertise required to trace errors to specific software modules and development teams when a defect is detected.

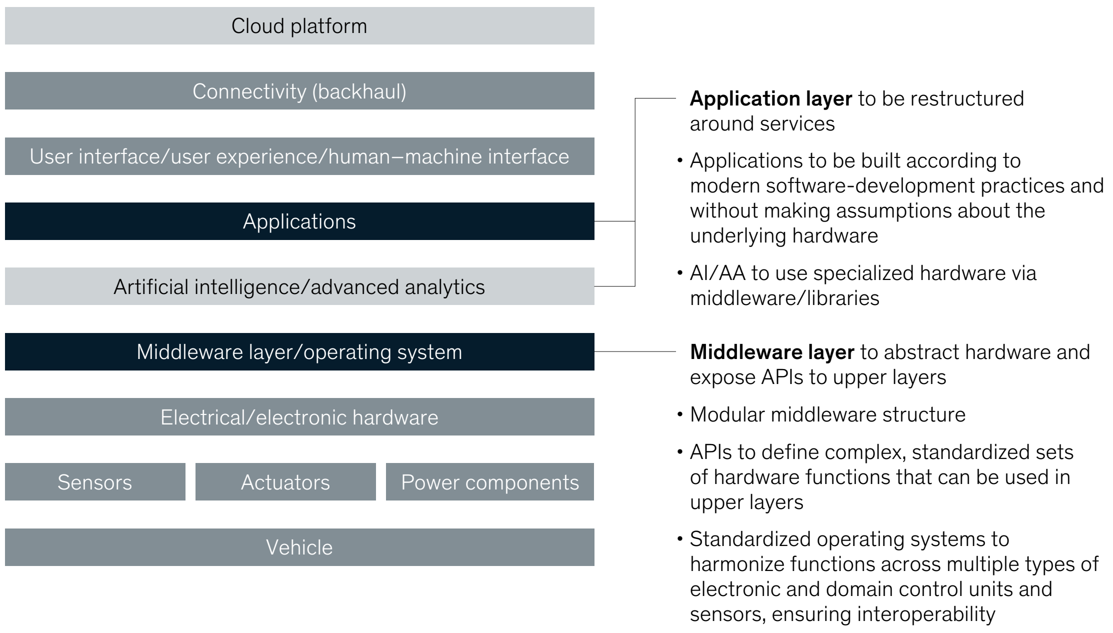
To address these issues, companies should drastically increase standardization and modularization, which can extend across platforms to keep software complexity manageable. OEMs must also focus on decoupling software from hardware and applying a service-oriented design.

Decoupled architecture. Companies can introduce a strong middleware layer that abstracts hardware capabilities and makes them available to software functions and services via standardized APIs used in upper layers (Exhibit 4). This software architecture can drive commonality across platforms and reduce design complexity, thereby eliminating the need to redevelop the same software multiple times.

In addition to decoupling, companies should also strive to create a set of standardized operating systems that support harmonization across components and ensure interoperability. Many players have already announced the development of such operating systems, but at this point, no

Exhibit 4

Companies should strive for a target architecture that supports decoupling of hardware and software and features a strong middleware layer.



Cloud platform
Connectivity (backhaul)
User interface/user experience/human–machine interface
Applications
Artificial intelligence/advanced analytics
Middleware layer/operating system
Electrical/electronic hardware
Sensors
Actuators
Power components
Vehicle
Application layer to be restructured around services
• Applications to be built according to modern software-development practices and without making assumptions about the underlying hardware
• AI/AA to use specialized hardware via middleware/libraries
Middleware layer to abstract hardware and expose APIs to upper layers
• Modular middleware structure
• APIs to define complex, standardized sets of hardware functions that can be used in upper layers
• Standardized operating systems to harmonize functions across multiple types of electronic and domain control units and sensors, ensuring interoperability

one-size-fits-all approach exists. And companies have yet to define the exact focus and functionality of these systems.

Players can effectively deal with increased system and software complexity by following clear architectural principles and guidelines. The hardware/software decoupling approach allows multiple entities to engage in modular development. In turn, modular software buildup techniques amplify code reuse and reduce the overall amount of code required due to increased commonalities. Many players have started to introduce software product line engineering (PLE) to increase reuse and handle product variation. This approach allows one piece of software to serve several products, product variants, and product families; it can also work in different hardware versions. Software PLE significantly decreases development and testing effort, since both only occur once.

Put another way, decoupling hardware from software further "democratizes" the hardware, which offers standard compute, memory, input/output, and power supply, while software defines end-user functionality. For applications requiring standard performance, different software functions can run on the same hardware using virtualization and containerization and be distributed dynamically to other hardware if necessary (for example, in the case of hardware failure). For applications with real-time performance requirements, such as ADAS, hardware-specific software development is still critical to achieve optimal efficiency.

Service-oriented architecture. The architecture should follow the definition of services, which, in turn, should codify business or user needs. A service-oriented architectural design allows companies to standardize core elements and interfaces across divisions and business units. Companies should also apply a standardized design for individual hardware and software elements so they can scale their resources, such as compute and storage, with other supported devices and functions without affecting performance. For OEMs, a service-oriented architecture is especially important. Achieving fast connectivity from vehicle to cloud will increase the long-term value of their models, as will rapid update and upgrade capabilities.

A2. Apply user-centered design techniques
Across industries, companies that concentrate on developing strong user designs and creating an optimal user experience (UX) achieve greater financial gains than others.[4] As ACES continue to gain traction and software-defined cars become the norm, these features will become increasingly important to an OEM's overall competitiveness. Even the top players will need to up their game, since the automotive industry still lags behind other sectors when it comes to designing a good software UX and providing optimal customer value.

Following best-practice design principles, OEMs should iterate new software offerings with end users, both before and after delivery. They should also adopt new delivery models that allow them to make software updates or additions on a weekly or monthly cadence, thereby allowing continuous improvement. These cycles are much shorter than those for classic hardware development, which typically take several years. As another benefit, the new delivery models will bring OEMs into direct and continuous contact with customers, allowing them to continuously receive feedback that helps them optimize requirements and provide a positive UX. These interactions were impossible when OEMs relied on hardware for upgrades, since contact occurred only during one-time sales through a dealer network or during market research. In order to get to this target state, OEMs need to implement required pre-prerequisites, such as a supporting software and electronic architecture, as well as toolchains to allow over-the-air updates.

Automotive OEMs that want to become UX leaders must use their data. With the increasing amount of in-vehicle software and sensors, automotive OEMs now have access to an enormous amount of information on how customers are using their vehicles. OEMs can mine these data to

[4] Based on analysis of correlation of McKinsey Design Index Scores with business development. For more, see Benedict Sheppard, Hugo Sarrazin, Garen Kouyoumjian, and Fabricio Dore, "The business value of design," *McKinsey Quarterly*, October 25, 2018, McKinsey.com.

Successful software development requires the continuous adjustment and correction of requirements based on customer feedback.

identify the features that are most important to customers, as well as those that are overspecified or not used at all. Such insights will inform the specification and prioritization of future model requirements.

Finally, the new delivery models will have a positive impact on development efficiency. Since OEMs will frequently change, adapt, and modify software, they will not need to specify extremely detailed requirements at project outset. With less time spent defining requirements, time to market can also decrease.

A3. Adapt software-requirements management
Historically, the automotive industry has been a leader in managing requirements in an integrated value chain. But OEMs primarily focused on hardware requirements, and their established processes are not optimally suited for software. With in-vehicle software becoming the major differentiator, OEMs must adopt new practices for managing requirements. The need for change is critical, since our research suggests that requirements for automotive software have become so detailed that they are slowing development.

Following best practices, OEMs should cluster requirements based on customer value. The first level should primarily include requirements that are customer facing (typically described as use cases). Technical or implementation requirements, such as the memory needed for a certain feature, should be in a separate level (typically described as enablers). This approach ensures that OEMs will focus on value creation and set the right priorities during software development. As companies divide requirements into levels, the following activities can help.

Linking requirements to strategy and customer value. Successful software development requires the continuous adjustment and correction of requirements based on feedback. While companies should initially derive software requirements from their business strategies and objectives, they should periodically make adjustments based on customer feedback and development progress.

Ensuring end-to-end traceability. By closely tracking requirements along the entire value chain, companies can avoid unnecessary effort and accelerate development. But they can only do so if their development processes and toolchains enable rigorous requirement traceability from definition to acceptance. This clarity will help companies maintain a clear understanding of the requirements (the customer's view), the needed functionality (the developer's view), and the deliverables (the tester's view).

Companies must enable end-to-end tracking through four steps that involve either a few highly integrated tools or four specialized tools with corresponding interfaces. The steps are:

— tracking requirements, which details and specifies the requirements from feature to component

— managing the backlog, which helps teams manage the coverage of requirements in software-development sprints (closely linked to the next step)

— tracking code changes, including updates of backlog items

— verifying requirements by undertaking test cases and checking the status of pass-fail test cases

By linking requirements through tools, users can efficiently implement changes in every phase of the project and thus fulfill regulatory end-to-end traceability requirements (for instance, ASPICE or UNECE[5]). This approach quickly makes it clear and transparent which changes affect what work products. When following an agile process, such requirement changes are normal and desirable (for instance, based on customer feedback) and companies should support them via processes and tools. In the traditional waterfall process for software development, these kinds of changes are rare and usually not foreseen.

Avoiding overspecification and creating clear categories. Companies can establish best practices to specify and categorize software requirements and develop simplified testing. A good requirement specification should be unambiguous, clear, and allow testing independent from other requirements. As with portfolio management, companies should distinguish between different types of requirements. Common categories include legal and regulatory, safety, strategic and essential improvements, customer value, and cost enablers. Furthermore, companies must ensure that any interdependencies between requirements are transparent. Many companies embed these rules in their software-development processes and training curriculums to optimize processing and review.

Prioritizing and continuously adjusting. Organizations should assess and prioritize software requirements based on their specific business cases and strategic goals, as well as customer feedback and learnings gained throughout the development phase (for instance, while testing). Companies should regularly reevaluate their requirements and maintain them in a transparent backlog.

Many companies appoint product owners with a broad knowledge base that allows them to evaluate trade-offs, bring cross-functional teams together, and ensure alignment among diverse functions about requirements. Product owners are also responsible for following best practices and maintaining the backlog of requirements and use cases.

B. Where software is developed: Organization, locations, talent, and partners

Most automakers lack the organizational underpinnings required to handle large-scale software development. Challenges range from having little or no executive-level responsibility for software to insufficient numbers of software engineers and designers. The new operating model will address these problems by defining the required organizational setup, location, and talent strategy for software development, as well as make-or-buy strategies and the required partnership ecosystem.

B1. Adapt the organization, and set up global centers of excellence

At the organizational level, most automotive players are not prepared to respond to the ACES trends that are increasing software's importance. For instance, OEMs tend to be slow when making decisions about software issues and many have not yet clarified the vehicle-platform overarching ownership of software, the electronics strategy, and associated budgets. To remain competitive in the new environment, companies must rethink their organizational setups. One major goal is to reduce the interfaces in architecture definition, requirements definition, and development during the end-to-end development process. By increasing alignment and collaboration among groups at all stages, companies can avoid

[5] Automotive Software Performance Improvement and Capability dEtermination; United Nations Economic Commission for Europe.

redundant efforts, and optimize both existing and new capabilities. Many companies have started to rely on a centralized function that takes responsibility for architecture and for sharing institutionalized best practices. For example, one OEM recently created a central function that brings together more than 5,000 software developers. In most cases, however, OEMs still take a more decentralized approach to software development.

Organizational archetypes. There are several common organizational archetypes that OEMs could adopt when attempting to improve software development. The best option for each company is the one that reflects its priorities, including those for accelerating decision making, reducing interfaces, and clarifying responsibilities. Even more important, companies must make a dedicated effort to harmonize requirements from different organizational functions, including R&D, procurement, production, sales, and aftermarket, as these may have conflicting requirements and life-cycles for software. This will help ensure a seamless user interface and an efficient development process. In addition to redesigning their organizations, companies should invest heavily in actions to close culture gaps and harmonize work processes. Such efforts will require sustained change management, but they will help automakers become software powerhouses.

When defining their organizational structure, automotive software-development units will ideally consider functional roles, products or projects, and technologies. The disciplinary organization structure will, however, follow one of these dimensions.

Consider an organizational structure that emphasizes functional roles. Under this model, product- or platform-specific projects are staffed with individual members from the functional organization. The product and technology dimensions will be represented through indirect, "dotted line" reporting into the functional units. This archetype gives organizations maximum flexibility, since they do not need to reorganize to accommodate further products/projects and technologies. Development efficiency under this archetype might be suboptimal, however, which will increase the need for cross-departmental functions, such as project management, architecture, and staffing.

In the second archetype, the organizational structure focuses on projects, such as those for specific customers, classes of vehicles, individual vehicles, and platforms. It connects to the product and technology dimensions via "dotted line" reporting lines and mature processes. This archetype makes the customer and the end-product the organization's primary focus. On the downside, it raises

While automotive organizations must excel on many levels to win the software game, attracting and retaining top talent is probably the most crucial dimension.

the risk of redundancies and creates barriers between different product or project groups that might interfere with technology transfer. Strong platform and architecture functions can help mitigate these risks.

In the third archetype, the organizational structure focuses on technologies and domains, such as the network, the human–machine interface, or the back end. Under this model, product-specific projects are staffed with individual members of the technology organization. This approach achieves the required focus on the technology and role dimensions through dotted lines and mature processes. While this model fosters deep technology and domain expertise, it provides little flexibility regarding project scope, requirements, and specifications, even if these change during the project. Organizations typically follow this archetype when they develop and maintain numerous products.

B2. Access and attract top talent and conduct in-house capability building

While automotive organizations must excel on many levels to win the software game, attracting and retaining top talent is probably the most crucial dimension.

Most OEMs have heavily outsourced their software-development activities and often rely on strategic partnerships. With ACES trends massively increasing the importance of software, and despite potential increases in software productivity, demand for software engineers will likely increase three to four times by 2030. Since the automotive sector is in direct competition with tech companies and other industries for software talent, it needs to take drastic steps to improve recruitment for top developers. Otherwise, the ever-increasing talent gap will continue to grow.

Automotive recruiting and retention programs should focus on achieving heterogeneity, since our benchmarks show that teams with a diverse mix of experience outperform homogenous teams by double digits in development productivity. Several activities can improve their talent efforts:

Increasing access to software-development talent on a global scale. A comprehensive location strategy can help companies scale their software-development activities, build relevant capabilities, and increase capacity while keeping costs in check. It will also help them compete for top talent. A few innovative companies have built new automotive engineering centers, such as those for autonomous driving, in digital-talent hot spots. Most traditional players have largely retained their historic footprints and hardware engineering centers, however, and this may complicate their efforts to attract and retain software talent. If these companies establish a global footprint with multiple locations in critical areas, they will gain access to talent from nearby universities and other educational institutions. To strengthen their connections, they could develop fellowship programs in partnership with the universities which would give them access to recent graduates and others with specialized skill sets.

While a global footprint confers multiple benefits, it also requires a suitable operating model to avoid common pitfalls arising from remote and/or distributed work. For example, research shows that software productivity erodes by an average of about 10 percent each time a new site is added to a development project.

Making the company more attractive to software talent. Our research shows that compensation, career-growth opportunities, type of work, and company culture are the top retention factors for software engineers. Currently, automotive players fall short in all these categories compared with tech companies. To close the gap, they must develop distinctive and targeted employer value propositions for all priority retention factors. They cannot simply emphasize traditional benefits, such as employment security and access to a company car.

Since the automotive sector is in direct competition with tech companies and other industries for software talent, it needs to take drastic steps to improve recruitment for top developers. Otherwise, the ever-increasing talent gap will continue to grow.

Encouraging in-house capability building. It makes sense to reskill current hardware-focused employees for software positions whenever possible. For instance, companies can train hardware project managers to oversee software projects. However, this practice only works well when companies maintain a good balance between software professionals and retrained hardware specialists. This balance might vary from company to company and is difficult to quantify. As a rule of thumb, however, a ratio of 2:1 for natives and retrainees has proved successful at many companies. For best results, companies should create attractive development plans and on-the-job training programs that help employees quickly learn new software languages and approaches. They should also emphasize lifelong learning.

Although retraining may fill many talent gaps, companies should remember that most people develop ingrained patterns for solving problems after they have been in a job for many years. When unexpected issues arise, software employees with a hardware background tend to apply internalized methods that are more likely waterfall-influenced than agile. Companies should thus try to encourage new ways of thinking during training. If they are successful, the differences between native software experts and retrained employees will quickly diminish. By contrast, companies that neglect this task could hinder the agile transition of their entire organization.

Providing career paths. Most automotive companies still fall short in defining career paths and growth opportunities compared with tech players. This gap occurs because specialist career paths are still not widely available at automakers. Furthermore, OEMs seldom define role expectations and seniority levels. Experts who want to advance are forced into management roles because they lack other options.

To improve retention, automakers can introduce clear career pathways that are linked to specific skills at each level. Some pathways may be for specialists while others are geared toward advancement. The latter may involve vertical advancement (from junior to senior developer to team lead) or equally critical horizontal job rotations, lasting six or more months, which focus on developing different skills, including those related to leadership, project management, and software development. Companies should also make dedicated training programs, including functional and interdisciplinary sessions, available to the broader organization. Clear career pathways may drive efficiency, since experts are typically more productive than novices.

B3. Define clear make-or-buy strategies and develop a partnership ecosystem

To keep hard-won competitive advantages and avoid becoming undifferentiated hardware platform developers, automakers must develop clear customer strategies, identify differentiating features and value propositions, and create modular architectures that can logically break up development modules. When these tasks are complete, they can create a clear make-or-buy strategy by focusing on three different dimensions (Exhibit 5):

— the phase of the development process, such as system integration or acceptance testing

— the software technology stack

— the software domain or module, which might cover areas such as infotainment or powertrain

Along these dimensions, companies should identify and define control points to position the make-or-buy strategy in line with their overall strategy (for example, for critical intellectual property, quality standards, and differentiating innovations).

Of course, companies must also consider the ease of sourcing from the market during make-versus-buy decisions. Other factors, such as cost, are also important but typically not decisive. To improve decision making, OEMs can weigh each factor based on their priorities and the market context.

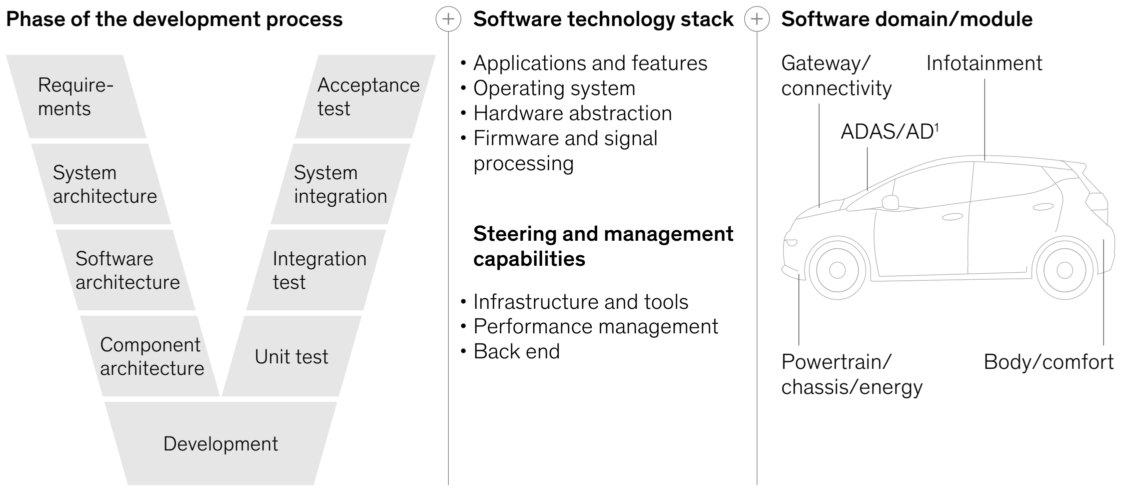
When a company decides to develop software in-house, it must assess the impact on internal engineering capacities, determine if current employees possess all necessary capabilities, and examine organizational structures and processes. If a company lacks the right capabilities or has insufficient capacity, it should explore acquisition opportunities or joint ventures that will allow it to maintain ownership of the critical control points.

If a company decides to buy software, it must define a detailed sourcing model during an extended assessment that involves selecting and contracting development partners. When considering a partial-buy strategy for a complex software system, companies should contract two to three suppliers

Exhibit 5

Companies should consider three dimensions during make-or-buy decisions.

Automotive example



[1]Advanced driver-assistance systems and automated driving.

Automakers have traditionally viewed software as secondary to hardware. They must now revisit this perspective, as well as their development approaches, since software is now a prime value driver in the product development portfolio.

at most. Our research shows that anything beyond that point can erode productivity by more than 65 percent.

In any comprehensive make-or-buy strategy, companies should use standard and open-source building blocks, since these can provide a huge advantage during software development. Companies will need to establish clear rules and processes for using open-source blocks, however, and pay careful attention to licensing, liability, and maintenance issues. Often, OEMs and suppliers will need a formal legal agreement to incorporate open-source components into a product.

Finally, automakers should develop strategic partnerships and identify ecosystem collaborators, since these connections allow companies to learn from each other while expediting development and keeping costs low. Codevelopment also reduces risks related to late market entry.

In a strong make-or-buy strategy, OEMs will keep production of differentiating features in-house while outsourcing development of noncritical software to other providers or contractors. Among other benefits, this approach will significantly reduce the demand for software talent.

C. How is software developed: Agile practices, decoupling, testing

Automakers have traditionally viewed software as secondary to hardware. They must now revisit this perspective, as well as their development approaches, since software is now a prime value driver in the product development portfolio. This shift will require automakers to implement agile-at-scale, decouple hardware- and software-development processes, and increase test automation and continuous integration.

C1. Implement agile-at-scale

Agile approaches, when applied at-scale in both hardware and software, allow companies to increase productivity and react quickly to changes in today's fast-paced environment. In agile transformations across industries, companies have achieved up to 30 percent increases in productivity and implementation speed while simultaneously reducing residual defects at time of release by over 70 percent.[6] By reducing project risks related to budgets, time frames, and quality, agile plays a critical role in mastering the complexity challenge.

To date, few automotive companies have consistently rolled out agile software-development practices. While many players are running pilots, especially in advanced development, only a few have implemented

[6] McKinsey's SoftCoster embedded software project database.

agile approaches at scale. The adoption rate may be low because automotive applications have very specific requirements that make it difficult to implement a standard agile approach across the organization. Further, the automotive agile tool set must be capable of handling system complexity, the often-complicated interdependencies with hardware development, and strict regulatory requirements concerning cybersecurity, vehicle safety, and quality.

Interdependencies with hardware development. To manage interdependencies, OEMs can adopt an agile-at-scale approach by combining systems engineering with requirements-management techniques. Together, those practices ensure smooth hardware-software integration, as well as a synchronized development timeline.

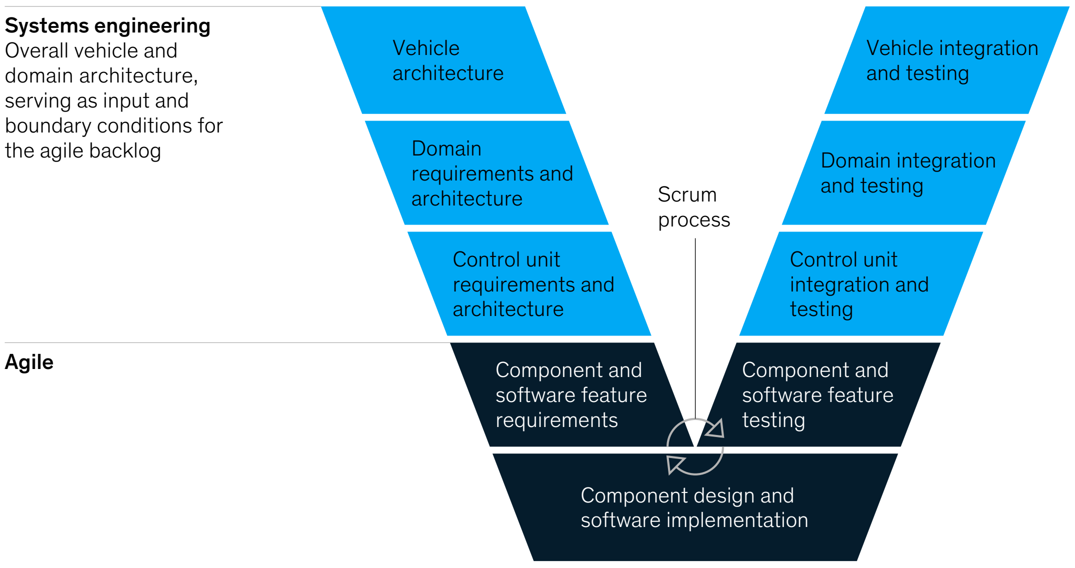
Classical systems-engineering practices for overall architecture, integration, and testing will ensure an integrated product life-cycle management approach and help companies meet regulatory requirements. Companies can use systems engineering practices to define the overall vehicle and domain architecture (Exhibit 6). This, in turn, will allow them to provide agile teams with high-level input and boundary conditions. Based on these inputs, agile teams can detail software requirements before developing and testing the components. At the end of the development cycle, the team will close the systems-engineering loop when domain, vehicle-integration, and testing activities bring the full system together.

From a project-management perspective, the goal is to gain functional alignment about the priorities and the required synchronization points for dedicated elements of the control unit and domain architecture. For instance, companies should prioritize and track the delivery of features required to start time-sensitive events, such as winter

Exhibit 6

Automotive companies should combine overarching systems engineering with agile software development.

Combination of systems engineering and agile


Systems engineering
Overall vehicle and domain architecture, serving as input and boundary conditions for the agile backlog
Agile
Vehicle architecture
Domain requirements and architecture
Control unit requirements and architecture
Component and software feature requirements
Component design and software implementation
Scrum process
Component and software feature testing
Control unit integration and testing
Domain integration and testing
Vehicle integration and testing

tests. Meanwhile, they would only monitor less critical features when looking at overall requirement-completion indicators.

Regulatory requirements and industry standards. Agile development practices work in automotive environments and unlock significant efficiencies, but companies must take regulations and the essential synchronization of hardware and software into account. As a result, they may need to change some of their traditional work processes, such as the approach for creating certification artifacts (for instance, to comply with the ISO 26262 standard). With classical waterfall development, certification artifacts are produced in the same order as engineering activities—planning, customer requirements, system architecture, software requirements, software implementation, software testing. With agile development, the best approach involves reverse certification—creating the certification artifacts at the end of the project, when all software requirements and code are stabilized, but before market launch. This process minimizes waste by ensuring that certification artifacts are not produced multiple times, and it requires software safety auditors to be well aligned and on board from the beginning of the project. Decoupling hardware and software may further simplify ISO 26262 certification efforts, since reused software might go through a compliance process via the proven-in-use argument, which can be applied when other applications have already used the software without incident.

Industry standards, such as ASPICE, currently mandate the traceability of all requirements and the ability to audit the processes and tools used. The traceability of requirements is compatible with agile practices and can be efficiently achieved with an automated toolchain (see section A3, requirements management, and D2, standardized toolchain). However, the need to audit processes and tools can limit the continuous improvement system inherent in agile techniques. While "pure agile" teams might improve their processes and working approaches independently, automotive teams must maintain their compliance with documented standards and synchronize across teams—actions that can slow their progress.

Agile development practices work in automotive environments and unlock significant efficiencies, but companies must take regulations and the essential synchronization of hardware and software into account.

When to use agile methods. Despite the need for predefined backlogs as well as auditable processes and tools, automotive software teams can readily adopt most agile practices. But doing so will dramatically change their working styles.

When shifting to agile methods, organizations must make critical design choices about macro-level operations, including portfolio, resource, and project management. Typically, only advanced development units, such as "software factories" at OEMs, follow an agile-at-scale approach in which all functions fully adopt new ways of working. There are exceptions, however. For instance, some automotive pioneers have implemented agile-at-scale methodologies, such as the scaled agile framework (SAFe), to steer their overarching R&D operations.

In contrast, individual teams should always follow established agile practices for operations. For instance, it is important to have cross-functional representation and members at the same location, as well as time-boxed iterations. As in other industries, the benefits of agility may be most apparent when applied to teams responsible for individual features (Exhibit 7).

As OEMs move to agile processes, they must also:

— Integrate development suppliers into their agile processes to enable full implementation

— Adapt procurement processes as they move from clearly predefined and prenegotiated specification-based contracts to a sprint-based development partnership

— Resolve legal issues that interfere with co-location strategies or agile collaborations between suppliers and OEMs

Exhibit 7

Agile methods can drive changes across multiple dimensions.

Example changes (not exhaustive)

Category	Example	From	To
Structure	Product team setup	Part-time staffing	Full-time staffing
		Separate teams for each function	Fully cross-functional
Ways of working, roles and responsibilities	Outcome-based steering/objectives	Activity-based	Outcome- or key performance indicator-based
		Measured against predefined project plan	Continuous evaluations
	Continuous release cadence	Waterfall, 3–4 times per year	Agile, weekly or daily
	Decision making	Hierarchical, at senior level	At level of product owner (what is developed) and agile team (how it is developed)
Technology and architecture	Modular or decoupled software architecture	Monolithic	Service-oriented
	Scalable IT infrastructure	Rigid architecture; sometimes patchy IT systems and incomplete data models	Modular IT architecture along with streamlined and scalable IT infrastructure

C2. Decouple hardware and software to enable two-speed development processes

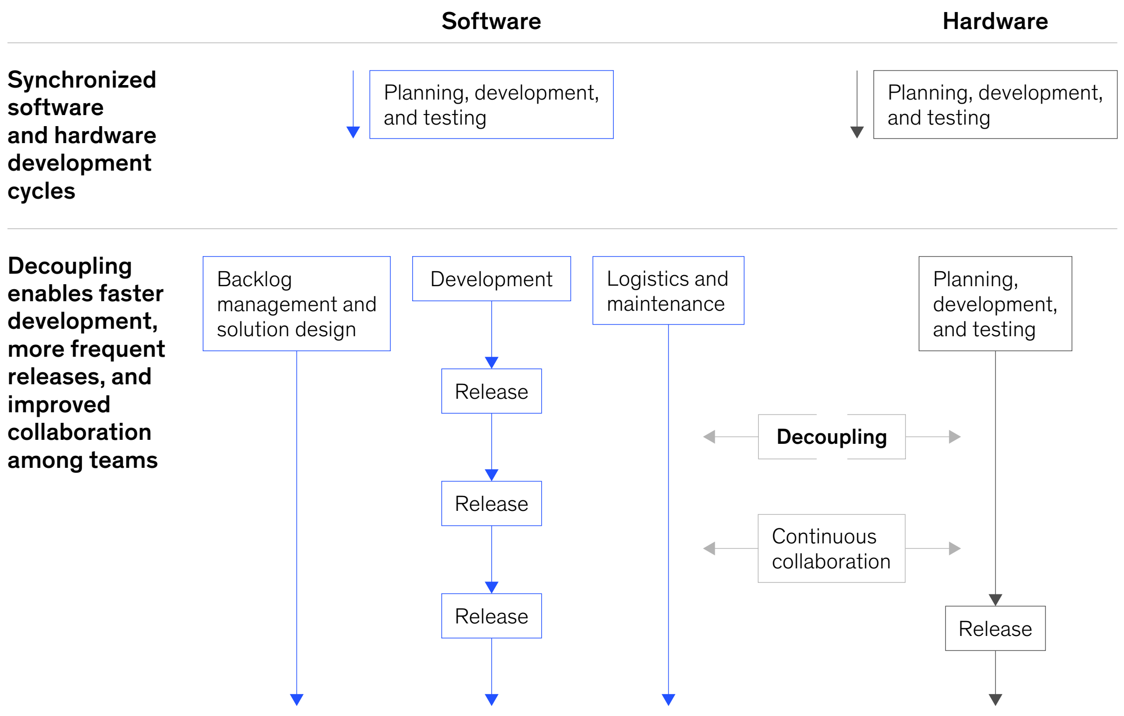
On the process side, automakers can pursue a more dynamic software-cycle plan that supports frequent releases that are not tied to rigid, distant vehicle-platform SOP dates (Exhibit 8). Decoupling product and life-cycle management from hardware is key to moving away from a one-vehicle SOP orientation. To do so, they must maintain separate backlogs and roadmaps while defining clear and synchronized milestones between hardware and software development. When working with vendors, OEMs may need to restructure agreements. Finally, they must intensify the use of automated software and integration testing and deployment.

As with the agile approach, a system-development team can manage and define the interfaces between hardware and software teams to split hardware/software backlogs and ensure synchronization across levels. Obviously, if hardware and software are independent, companies can separate architecture freeze points and will thus have no need for synchronization.

Like agile methods, decoupling requires several prerequisites, including a standardized and modularized architecture and a robust test approach. As noted earlier, companies can decouple software from hardware by using a strong middleware layer that abstracts hardware capabilities and makes them available to functions and services through standardized APIs. However, the most critical enabler for decoupling processes is a robust approach with test vehicles. Consequently, companies must define the practices for providing and maintaining

Exhibit 8

Companies gain many advantages by decoupling software and hardware development cycles.


Software
Hardware
Synchronized software and hardware development cycles
Planning, development, and testing
Planning, development, and testing
Decoupling enables faster development, more frequent releases, and improved collaboration among teams
Backlog management and solution design
Development
Release
Release
Release
Logistics and maintenance
Decoupling
Continuous collaboration
Planning, development, and testing
Release

the test vehicles: either hardware-in-the-loop or software-in-the-loop systems or the broader simulation infrastructure.

Both agile methods and decoupled hardware/software development have significant implications downstream in the value chain, especially for the procurement organization. For instance, procurement will need to shift from a traditional waterfall-based sourcing process to more agile and decoupled development approaches. In practice, this requires OEMs to follow certain best practices, such as continuously assessing the strategic importance of software elements, understanding how requirements will evolve, and determining which sourcing models are most appropriate in each instance. These changes will require a total-cost-of-ownership perspective on software, as well as new cooperation models that focus on strategic partnerships instead of multisourcing.

C3. Increase test automation and mature continuous integration to ensure the earliest possible error detection

With software volumes increasing and greater demand for continuous feature updates, OEMs must find and resolve bugs and interface errors as soon as possible. If they fail to resolve an error directly, they may create a massive backlog of bugs that could fully absorb their resources and complicate error tracing, causing development delays and increased verification efforts.

As with agile practices, few automotive players have adopted continuous integration or automated testing practices at scale. If they reverse this trend, they could simultaneously reduce launch risk while dramatically increasing productivity. In our experience, companies have increased productivity by over 40 percent while reducing residual defect density by more than 60 percent.

To follow their examples, automotive companies should adopt two interrelated software-development best practices. First, they should integrate code into a shared repository several times a day and verify it via an automated build. By integrating code early, developers can “fail fast” and easily isolate errors through continuous integration practices, tooling, and the use of automation. Suppliers may rather independently realize these benefits on a system level. On a full vehicle level, OEMs need to overcome IP constraints and either insource coding or achieve a “whitebox” approach of full code sharing with the supplier. The second element of the solution is test-driven development and automation, a process in which tests are defined before coding starts and then automatically run after code integration. Software designers and developers—not a separate testing department—refine and iterate the tests during the development process with customers. This approach compels developers to consider how to use a system and how to implement it before coding. Over time, a comprehensive automated test suite will enable sustainable, high-quality sprints.

D. How is software development enabled: Performance and toolchains

To optimize efficiency from a software-driven operating model, companies should adopt a best-in-class performance-management approach and set up state-of-the-art software infrastructure by building a software-development toolchain.

D1. Implement performance management that considers data-driven productivity, project maturity, and quality measurement

As software complexity increases, automotive players must upgrade their performance-management systems using standardized, data-driven metrics for productivity, project maturity, and quality. Only automated, data-driven insights can enable a real-time, fact-based performance-management approach and proactively reveal looming software issues concerning time, cost, and quality.

A best-in-class software performance-management system will excel across three key activities:

— Establishing a comprehensive system that includes cascaded key performance indicators (KPIs), such as those for cost, time, and quality, and ensures that each one links to an overarching business target and operational tasks

The introduction of a standardized, state-of-the-art development toolchain is a key enabler to unlock 30 to 40 percent of productivity potentials from automated testing and agile methods.

- Developing an efficient system for escalating issues that includes standard meetings focused on simple and quick reporting, decision making, and escalation

- Using tools, automated productivity-measurement technology, and code quality metrics to ensure objective productivity and quality measurement for software development

D2. Upgrade development toolchains
Automotive players can increase efficiency by introducing a standardized, effective software-development toolchain that supports continuous integration and the use of standard APIs. Typical components of this chain include source-code management processes and tools focused on build, continuous integration, and test automation (test execution, test-verdict generation, and test-report generation). As noted above, this toolchain also seamlessly integrates all tools for requirements management.

Building an integrated and highly automated toolchain can unlock significant benefits during the development process. For example, it increases internal efficiencies by reducing complexity and enables the use of established technology from external building blocks. Based on our research and experience, a state-of-the-art toolchain can:

- Drive faster and more agile development, lowering the effort required for code releases or builds by 90 percent

- Reduce defect density by enabling use of strict quality gates, potentially reducing failure rates by 50 percent

- Allow companies to compile thousands of code builds per day, without any manual effort

Overall, the introduction of a standardized, state-of-the-art development toolchain is a key enabler to unlock 30 to 40 percent of productivity potentials from automated testing and agile methods. This level of performance differentiates top performers from the pack. Automotive players should view this type of software-development toolchain as the backbone for a highly productive organization that supports continuous integration and the use of standard APIs. Such toolchains also ensure efficient and automated interfaces between software and hardware development tools throughout the full development process. They also create an opportunity to automate several process steps for activities such as test runs. Overall, the goal is to accelerate development speed and enable early testing.

Often, the number of toolchains and tools used will increase to unmanageable levels, reducing transparency. To avoid this problem, experienced toolchain managers should regularly review the tool landscape and take corrective actions when needed.

Making the transformation happen

Many organizations are already taking serious steps to capture value from software and thus benefit from the opportunities offered by the ACES trends. Typical transformation journeys start by focusing on a single challenge or topic, such as immediate remedies in software project recovery, software make-or-buy decisions, or a desired step-change in software-development performance. Most companies are discovering, however, that a single intervention has limited impact, especially if they lack the critical foundations required to sustain improvement. To set up world-class software-development capabilities, firms should therefore take an end-to-end transformation view. Depending on their individual starting points, companies may approach transformations in different ways. When choosing an initial focus, they can select one of the following entry points:

‘What’: Product architecture and make or buy
The aim is to set up a target software architecture with both specific internal control points and a partnership ecosystem, allowing for efficient and competitive software delivery across platforms. (See sections A and B3)

‘How’: Productivity boosts and software-development methodology
The focus is on improving software R&D productivity by using a combination of key efficiency levers for software development, including agile R&D, continuous integration, and automated testing (see sections C and D2).

‘Where’: Talent access at reasonable costs
Companies at risk of having insufficient capacity to increase their software output may initially focus on this topic. Their solutions will involve optimizing both talent access and R&D costs by improving their footprints and making their organizations more attractive to software talent (see section B2).

‘How to set up’: Organization and governance
Another starting point involves defining the optimal organizational structure, determining “boxes and lines,” and specifying the operational model, including steering and performance management, to strengthen software delivery (see sections B1 and D1).

Overcoming the automotive industry’s current software complexity and productivity conundrum requires a comprehensive transformation of automotive software R&D. CTOs and CEOs must accept this challenge as a top priority on their agendas—and address it now to remain competitive and successful in the current industry environment, and they should prepare for an extended journey. Their transformations will take several years to address all issues related to a software organization and its underlying operating model.

Ondrej Burkacky is a partner in McKinsey’s Munich office, where **Dominik Hepp** is an associate partner; **Johannes Deichmann** is a partner in the Stuttgart office; **Stefan Frank** is a consultant in the Hamburg office; and **André Rocha** is a partner in the Tokyo office.

The authors wish to thank Silviu Apostu, Georg Doll, Julia Gößwein, Anna Gomulec, Virginia Herbst, Shannon Johnston, Maximilian Lemmens, and Henrik Rochlitz for their contributions to this article.