McKinsey & Company

Automotive & Assembly Practice

Software 'should costing': A new procurement tool for automotive companies

Analytics-based cost algorithms and agile-inspired methods can help OEMs reduce automotive software costs by up to 30 percent and improve delivery timelines.

This article was written collaboratively by McKinsey's Advanced Electronics Practice. The authors include Roberto Argolini, Ondrej Burkacky, Shannon Johnston, Stefania Pellegrinelli, and Georg Wachter.

September 2020

Faster, cheaper, better. Those are the words that come to mind when thinking about "should costing." This method, which incorporates both digital technologies and agile principles, involves using clean-sheet techniques to create a bottom-up estimate of a supplier's production costs and margins. As its name suggests, should costing helps companies distinguish between the set price of goods or services and their true value.

The should-costing method is now the gold standard for hardware purchases at automotive companies, but largely absent from software procurement. While this oversight may seem surprising, the explanation is simple: most automotive procurement groups are familiar with hardware suppliers and their work processes but understand relatively little about software development. Those companies that can overcome this knowledge gap and apply should costing to their software projects stand to win big, since experience suggests they can potentially reduce costs by up to 30 percent while simultaneously improving delivery time frames.

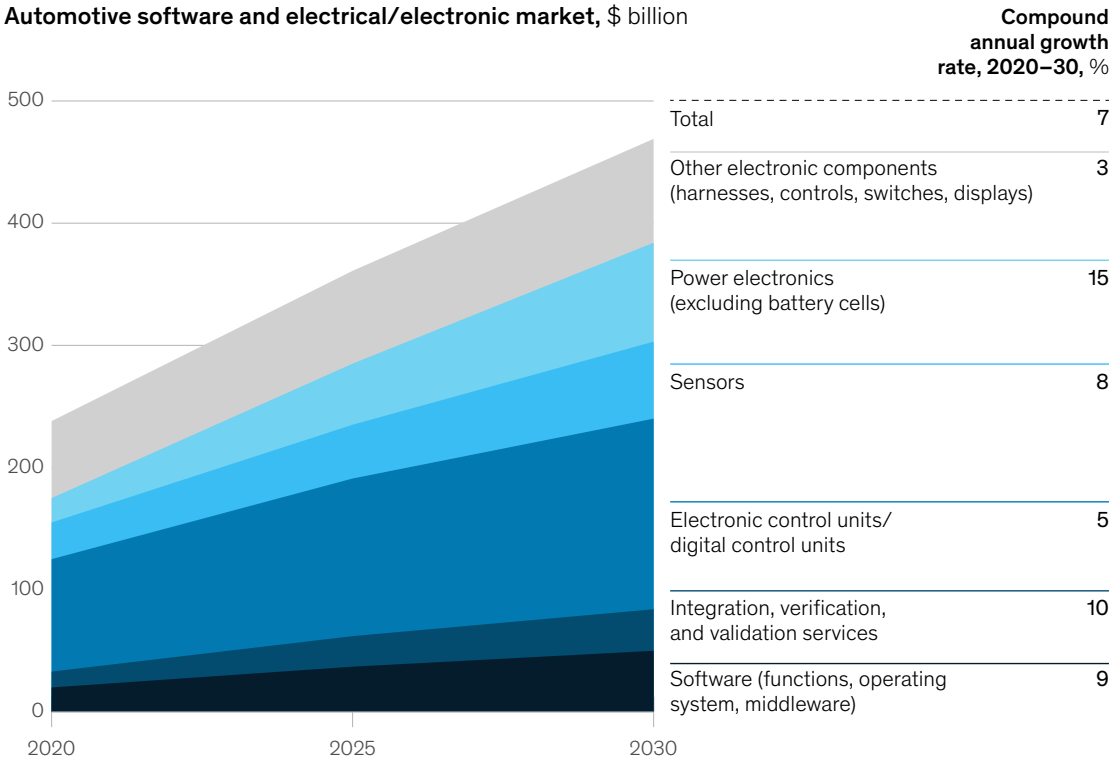
The growing importance of automotive software

As new technologies disrupt the automotive industry, software is becoming progressively more important. OEMs and other stakeholders increasingly view it as a key value driver for cars—one that is equally or more critical than traditional areas, such as power train and performance. Software that enables autonomous vehicles, connectivity, electrification, and shared mobility—often termed ACES—can strongly influence brand decisions. For example, McKinsey research shows that 36 percent of customers would willingly change brands for better digital and connected services.

In line with these developments, the global market for automotive software will double from 2020 to 2030. This growth easily outpaces expansion in the automotive market as a whole, which is expected to increase at a little over 3 percent annually during this period (Exhibit 1).

Exhibit 1

Strong growth is expected for the automotive software and electrical/electronic market.



Source: McKinsey analysis

The challenge of automotive software procurement

Although OEMs want to capture vehicle control points and develop at least a portion of their differentiating software in-house, most still rely heavily on external hardware and software suppliers to provide customized solutions. Software that enables some critical vehicle functions, such as connectivity and battery management, is often sourced externally. Likewise, software pure-play companies frequently design algorithms for vehicle systems, including machine-learning programs that control autonomous driving. The situation is similar at traditional tier-1 and tier-2 suppliers, who often draw on the expertise of their own software vendors when serving traditional automotive companies.

The same reliance on external sources also holds true when OEMs are looking for customized software to run different areas of their businesses—for instance, AI-powered, vision-based quality-control systems for manufacturing or pricing and forecasting algorithms for sales. Among suppliers, the ongoing digital transformation is prompting more IT departments to outsource a larger share of their software development because they are struggling to meet increased demand from the business side. This pattern holds true even among traditionally cost-focused IT functions.

Shortcomings during automotive-software procurement negotiations

Automotive stakeholders may be buying more software, but their procurement groups are still getting up to speed in this area. They have far to go before they can deliver a fact-based view of software costs because they do not have the tools and capabilities required to create an objective, accurate view of what products and services should cost.

Software remains a mysterious black box for procurement groups who often struggle with cost negotiations because they lack complete knowledge of suppliers and their development processes. They have few predeveloped best practices that they can apply, since most advanced, high-tech players typically produce most of their software in-house, on a proprietary basis. By contrast, they have hardware experts with in-depth understanding of suppliers, including their manufacturing techniques, and can easily estimate costs to specific goods and services.

Most automotive procurement groups are familiar with hardware suppliers and their work processes but understand relatively little about software development.

The widespread benefits of should costing for automotive software

The lack of transparency about software development and associated costs can have major repercussions. First, and perhaps most important, it often leads automakers and tier-1 suppliers to pay too much for code. An estimated 10 to 30 percent gap exists between the best-offered cost and the should-costing figure (Exhibit 2). The value at stake can amount to hundreds of millions of dollars for a typical OEM or large tier-1 supplier.

Software should costing, which is the best way to attain complete transparency, relies on a clean-sheet, bottom-up model of the production process to estimate a supplier's costs and margins. It helps turn software development from a single black box to a gray one and ultimately to a set of small, individual, full-color elements that clearly show all relevant cost drivers and deliverables. With this level of detail, buyers can move from a purely commercial discussion with suppliers and undertake a fact-based, point-by-point examination of the goods and services under negotiation.

Among other benefits, the should-costing method can help companies understand the financial impact of various deal elements, such as suggested team size, colocation, project duration, and the share of the workforce located in best-cost countries. For products, it allows companies to see how different feature sets or solution architectures will affect costs. Ultimately, the should-costing method also helps de-risk software delivery and product launches by allowing procurement groups to answer important questions, including:

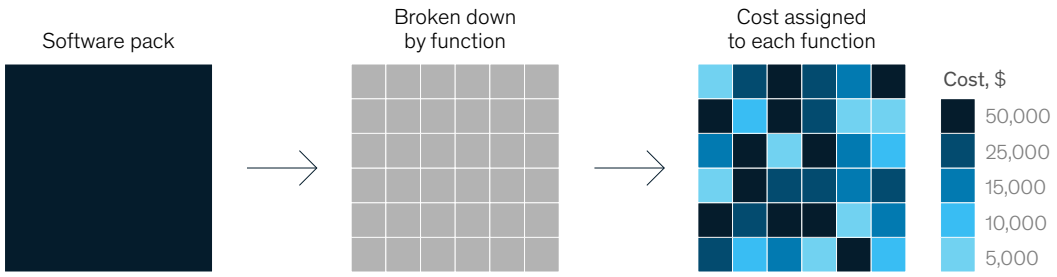
- How do we evaluate the actual competitiveness of a single offer?
- Are we receiving the maximum service for our money?
- How can we tell if the productivity and cost performance of our vendors are competitive, and how can we improve them?

These questions are becoming even more relevant as software increases in complexity and development requires many engineers and advanced skills.

Exhibit 2

Should-costing methods can reduce software costs by 10 to 30 percent.

How software should costing works



Should-costing benefits:

Improves negotiations through fact-based conversations on product price components	Informs knowledgeable discussion of supplier's team size, co-location plans, and project duration	De-risks software development by enhancing control and allowing OEMs to challenge the supplier during development	Helps to create interoperability regarding different feature sets or solution architectures

Gaining a shared, detailed understanding of software-development costs allows OEMs and suppliers to develop and grow together. It is the first step in a collaborative journey where cost drivers are not just negotiation points but common problems that both companies will tackle, step by step.

The options for should costing

When purchasing software, most OEM procurement departments simply request detailed proposals from suppliers that break down costs, including those associated with individual features or specific team roles, such as architects, developers, and testers (Exhibit 3). Purchasing managers then compare the various offers and use them to negotiate a "best of best" feature set and price across different suppliers. While this cost approach provides an effective basis for negotiations and supplier-performance management, it does not convey a true understanding of cost drivers.

As they move to should costing, OEMs can select from two methods: the "T-shirt sizing" approach and the complexity-point analysis.

Estimating the cost of software via the 'T-shirt sizing' approach

The fastest and easiest should-costing method involves applying a structured engineering technique frequently used during agile development sprints. This approach is common at many tech companies and involves asking experts to make relative estimates, rather than absolute estimates, because people are more likely to understand their significance. Since relative estimates may be difficult to visualize in relation to software, consider how the same process might work with clothes sizing. When looking at an extra-large T-shirt , people might not appreciate how big it is. But the extent of its volume becomes obvious when it is placed next to an extra-small T-shirt.

Exhibit 3

Three options exist for custom software cost-planning and procurement negotiations.

Method comparison

HIGH ← **Ease/frequence of use** → LOW

	RfP[1] size/quote comparison (Current method)	T-shirt sizing (Should-costing method)	Machine-learning-based, complexity-point-analysis approach (Should-costing method)
Description	Request a detailed cost breakdown for individual features/roles along the project timeline; benchmark/ compare different suppliers	Identify software building blocks (eg, functions); estimate effort and cost for each building block via the T-shirt size method	Define the software structure of the main functionalities/ sub-functionalities or detailed software functions with their pseudocode; estimate key complexity drivers for identified functionalities; estimate effort and cost with a machine-learning-powered comparison of projects in a reference database
Advantages	Easy to apply	Easy to apply	Limited effort; precise, top-down approach with objective results, including project/resource plans
Disadvantages	Imprecise approach; only provides relative results	Rough result; relative result; difficult to benchmark	Analytics-driven black box

[1]Request for proposals.

Should-costing techniques, already used for hardware purchasing, must expand to cover software purchasing to help companies obtain the best value for their money.

When applying the T-shirt sizing approach to high-complexity custom software, expert teams first break the software into manageable parts—for example, looking at the software by function or customer requirement. Second, the teams define size, following the T-shirt approach, by comparing the parts to reference projects. The teams typically include about six steps, from extra-small to extra-extra-large, often using that same terminology. The estimates follow the relative size of a Fibonacci series[1] to offset the uncertainties inherent in larger projects.

Finally, they convert the size estimate into a cost value by assigning one T-shirt size an absolute value (for instance, in man hours or dollars), typically based on historical in-house projects.

One automotive company created a T-shirt sizing model for a common electronic control unit (ECU). By breaking down the application and base software layers into individual parts, the team was able to have a detailed discussion about the full or partial re-use of individual features in the next-generation ECU. This strategy helped reduce development costs by over 25 percent from the initial estimates.

While T-shirt sizing has the advantage of speed, it does not allow teams to make comparisons across different projects, including those run by other expert groups. T-shirt sizing also uses a limited set of projects for comparison, and the resulting estimates do not typically include any effects related to schedule constraints or best-cost location.

Machine-learning-based complexity-point analysis approach

In a complexity-point analysis, companies use a set of reference projects against which they make standardized comparisons. With very complex and large software applications that take years of development and numerous engineers, the process first breaks the software down into blocks of manageable complexity. Typically, the blocks are based on functionalities of customer requirements and communication needs.

For each building block, teams estimate the key complexity drivers, which typically include the number of required tests, variants, and lines of code as well as the code type (for instance, new versus legacy). Teams also consider nonfunctional requirements, such as safety.

For smaller and more limited software applications, companies can improve their initial estimate by generating detailed pseudocode and data structures to calculate the number of functional points. They then correlate this information with the needs of overall system features, such as real-time requirements or parallel computation.

After these steps, players can convert the identified effort drivers into the actual cost and effort (for instance, engineering hours) required to deliver the project through comparisons with a set of relevant reference projects.

[1] A series of numbers starting with 0 and 1 in which each number is the sum of the two earlier numbers.

The most advanced complexity-point analyses employ machine-learning algorithms to select the best reference projects for the comparison, based on available information. For example, one McKinsey solution involves having teams estimate the total effort required by breaking it down into a project plan that includes staffing needs in various roles over time. The teams also provide a total cost estimate, including those associated with trade-offs, and compare them to those listed in a large database of software projects. For instance, teams might consider whether having all team members in one location would decrease the number of employees in best-cost countries, thereby raising project costs.

A complexity-point analysis typically requires a limited effort, although that can vary depending on type of inputs, but it provides a precise and objective result comparable across projects. This analysis also allows companies to set constraints, such as a project start and end date, the number of simultaneous necessary resources, and staffing requirements. All these factors can influence team productivity and are important to consider when conducting fact-based negotiations with suppliers to lower costs and optimize project execution.

When estimating the chances of a project's success compared to reference projects, or when conducting scenario analyses, teams can adjust project constraints. For example, they can determine how accelerating the timeline would increase project costs, or look at trade-offs between the value of individual software features and their respective development costs.

The increasing importance of custom, high-complexity software for automotive OEMs and suppliers is compelling procurement departments to monitor software spending more closely and improve their toolboxes. They must go far beyond basic comparisons of requests for proposal to find the best price. Should-costing techniques, already well established for hardware purchasing, must expand to cover software purchasing to help companies obtain the best value for their money. OEMs and tier-1 suppliers late to this party may soon regret it.

Roberto Argolini is an associate partner in McKinsey's Milan office, where **Stefania Pellegrinelli** is an expert; **Ondrej Burkacky** is a partner in the Munich office; **Shannon Johnston** is an expert in the Toronto office; and **Georg Wachter** is a consultant in the Vienna office.

The authors wish to thank Georg Doll, Mauro Erriquez, Dominik Hepp, and Alfredo Vaghi for their contributions to this article.

Designed by McKinsey Global Publishing