2019 IEEE International Conference on Software Architecture Companion (ICSA-C)

Continuous Software Engineering of Innovative Automotive Functions: An Industrial Perspective

Philipp Obergfell*, Stefan Kugele†, Christoph Segler*, Alois Knoll†, and Eric Sax‡
*BMW Group Research, New Technologies, Innovations, Garching bei München, Germany
†Technical University of Munich, Department of Informatics, Garching bei München, Germany
‡Karlsruhe Institute of Technology, Department of Electrical Engineering and Information Technologies, Karlsruhe, Germany
Email: {philipp.obergfell, christoph.segler}@bmwgroup.com, stefan.kugele@tum.de, knoll@in.tum.de, eric.sax@kit.edu

***Abstract*—One major challenge in the automotive industry is to deliver innovative functions more frequently. Hence, the standard development process with a fixed release plan is likely to be turned into a more continuous procedure. From a methodological perspective, this change includes applying well-established agile development techniques. In contrast to pure software-related domains, the successful implementation of software-based functions in embedded systems highly depends on non-functional requirements, and therefore we see the need for extending the pure code-centric notion of agility. In order to do so, we reflect on architectural drivers that are beneficial for the transformation of OEMs into software companies. Finally, we present our perspective on future automotive software engineering by illustrating how continuous integration is applied by a software engineer not only on the level of source code, but also on the system architecture level and the respective role.**

***Index Terms*—automotive, agile, process, continuous integration, continuous delivery**

I. Introduction

During the last decades, thousands of mostly software-controlled functions were included in modern cars, which are executed on a large number of electronic control units (ECU). Their particular characteristics reach from non-safety-critical to safety-critical and real-time-critical functions. Driving forces for this development are: (i) safety requirements, (ii) customer demands for more comfort and the newest infotainment systems, and (iii) advanced driver assistance systems allowing to reach higher levels of driving automation.

Current E/E Architectures: The resulting vehicle electric/electronic (E/E) architectures are best characterised as historically grown, mostly federated, partly integrated architectures with often pragmatic, cost-efficient, and ad-hoc solutions. A lot of research effort is currently put in developing all-new E/E architectures that will be even better equipped for future trends, innovations, and new technologies [1], [2].

Customer Demands: Machine learning is an emerging and cross-disciplinary area, which is advancing with great strides also for in-vehicle functions. Sophisticated *smart capabilities* and highly personalised functions are no longer a future vision but can already be found today. Rooted in the consumer electronics (CE) industry, customers demand new levels of *personalisation* (and individualisation). This extends the traditional aspects of personalisation for vehicles such as colour, interior equipment, or options by far. Customers expect innovations to be delivered instantaneously.

Continuous Updates: Regulatory authorities will by law require to improve, i. e., (i) fix possible errors and (ii) update crucial components to the state-of-the-art. Moreover, vehicles are in use for more than 15 years on average. During that time-span, information technology improves rapidly: E. g. an encryption algorithm used for backend communication can become outdated and not be considered secure anymore and has to be updated after almost a decade. Hence, possible attack surfaces need to be mitigated continuously.

Continuous Software Engineering: Customer demands, innovations, and competitors push OEMs to fast and lightweight software updates, which has to be enabled by fast-moving development processes based on the idea of continuous software engineering [3]. *Continuous Integration and Delivery* (CI/CD) help to improve the quality and speed at which automotive software-based innovations are delivered to the customers' vehicles—going along with the DevOps way of thinking. Such an approach requires the currently mostly rigorous V-model-based development to turn into a highly *agile* process known from modern software development methodologies. Note that today software delivery after the start of production is minimal.

Outline: This position paper focusses on the *development processes* and methodologies needed to deliver new functionality to customer vehicles quickly. Therefore, we first focus on the notion of architecture and possible drivers that foster agility. Second, we outline a process concept for continuous automotive software engineering from an industrial perspective.

II. Architectural Drivers

Besides automotive process models—like the stage-gate approach, which contradict the idea of incremental and continuous software engineering—we believe that especially architectural questions decide whether automotive software development can speed up. Having this in mind, we provide a set of architectural key drivers that demand the implementation of suitable technologies in order to foster fast change.

a) IP-Based E/E Architectures: Current automotive software architectures follow mainly the idea of static communication in line with the AUTOSAR Classic Platform [4]. As a result, changing the applicative parts of software leads in most cases to a change in the so-called communication matrix describing statically bound data channels between hardware senders and receivers. To overcome this, automotive OEMs are on the way to steadily substitute legacy communication by including more and more IP-based communication technologies into their vehicles, which in turn support a better separation of software and hardware.

b) Extending E/E Architectures: As restricted hardware resources constrain embedded software development, we pose the requirement of extendibility (e. g. computing resources) that needs to be accomplished when targeting continuous software engineering. • *Vehicle-Internal Extendibility:* Vehicle-internal extendibility of computing resources reflects trends in chip design for embedded systems. Briefly spoken, the concept of a microcontroller with classic single core CPUs needs to be substituted at scale by more powerful multi-core architectures. Moreover, their integration into extendible housings supporting future after-sales concepts by the OEMs needs to be reached. • *Vehicle-External Extendibility:* Vehicle-external extendibility of computing resources reflects trends in outsourcing software applications into backend systems or retrieving sensor data from surrounding vehicles/infrastructure instead of implementing the corresponding components internally. For this, a provision of real-time capable and secure interfaces between the vehicle and its environment need to be integrated.

c) Learning E/E Architectures: With the advance of machine learning algorithms, function developers are more and more developing data-driven functionalities, which can capture hidden knowledge for continuously improving its capabilities (e. g. [5]). The magnitude of built-in sensors holds the opportunity that required data has already been measured and could be directly used for data-driven functionalities without adding any additional hardware. However, this magnitude of sensors also leads to high complexity in current architectures and sensors are often hidden for the data scientist and not being used, even though if necessary. Additionally, due to the static nature of current architectures and the dynamic nature of learning systems, these systems cannot be fully integrated into current architectures. In consequence, the task of transitioning from static heterogeneous system design to dynamic central system design is inevitable.

III. Process

Based on posed architectural drivers, we now present our idea on future automotive software engineering. As a concept, two conjoined continuous integration pipelines (one for the architectural integration of new functionality and one for its realisation in software) are introduced.

a) Continuous Integration: Architecture Level: On the architectural level, our process depicts system design activities that are initialised by introducing new functionalities. The ultimate goal is to identify suitable computing resources within and outside the vehicle as well as to derive communication interfaces (network routing) in the case of distributed deployments. Both steps can be achieved iteratively. This is especially true when considering the product before its initial release to the customer. In the opposite case, changing a pre-existing system design is also considerable, but requires additional after-sales processes that are not depicted here.

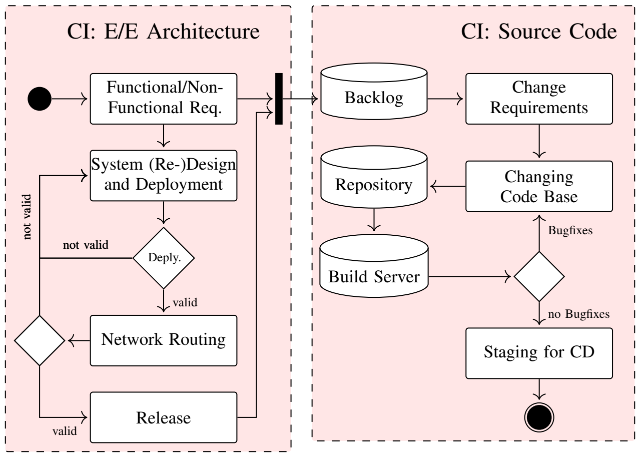

Fig. 1. Process

b) Continuous Integration: Source Code Level: Based on architectural change approvals within our first pipeline, functional requirements are released for implementation. To do so, the second part of our pipeline includes automotive tailoring of standard elements for continuous integration (e. g. build servers and repositories for change requests and source code). Since the current collaboration model between OEMs and suppliers is based on written specification documents, we strongly focus on the inclusion of suppliers and third-party contributors within the development and test procedures. After approval, mainly worked out by build and test scripts, staging and entering the continuous delivery pipeline takes place.

IV. Conclusion and Future Work

The idea of vehicles as computers on wheels is not new. However, on the level of design processes, approaches (mainly known from mechanical engineering) are still dominant and come true by fixed release plans and long development cycles. In this paper, we stated our idea on continuous automotive software engineering, which emphasises that agile software development is only beneficial when applying its techniques on the architectural level, too. For future work, we aim to develop and implement a corresponding pipeline that supports architects in an agile development environment and helps to speed up with innovation.

References

[1] M. Traub, A. Maier, and K. L. Barbehon, "Future automotive architecture and the impact of IT trends," *IEEE Software*, vol. 34, no. 3, pp. 27–32, 2017.
[2] S. Kugele, V. Cebotari, M. Gleirscher, M. H. Farzaneh, C. Segler, S. Shafaei, H.-J. Vögel, F. Bauer, A. Knoll, D. Marmsoler, and H.-U. Michel, "Research Challenges for a Future-Proof E/E Architecture – A Project Statement," in *15. Workshop Automotive Softw. Eng.* LNI, 2017.
[3] J. Bosch, Ed., *Continuous Software Engineering*. Springer, 2014.
[4] "AUTOSAR: Automotive Open System Architecture," http://www.autosar.org.
[5] P. Obergfell, C. Segler, E. Sax, and A. Knoll, "Synchronization between Run-Time and Design-Time view of Context-Aware automotive system architectures," in *2018 IEEE International Systems Engineering Symposium (ISSE)*. Rome, Italy: IEEE, Sep. 2018.