Chapter 9
Current Trends in Automotive Software Architectures

Abstract Cars have evolved a lot since their introduction and will evolve even more. Today's cars would not work without the software that is embedded in their electronics. Although the physical processes are often the same as in the cars' of the 1990s (combustion engines, servo steering), they become computer platforms and are able to "think" and drive autonomously. In this chapter we look into a few trends which shape automotive software engineering—autonomous driving, self-* systems, big data and new software engineering paradigms. We look into how these trends can shape the future of automotive software engineering.

9.1 Introduction

Automotive software evolves over time and requires changes to the methods used to develop it. The evolution of software means that we can use new functions which require more software, but also that we can use more advanced software development methods.

If we look at the history of electronics and software in cars, we can see that it is today that the big technological breakthroughs are happening. The cars of today have become sophisticated computer platforms which can be used in multiple ways. The powertrain technology has changed from traditional combustion engines to electrical or hybrid (e.g. hydrogen technology).

Living in these interesting times, software engineers and architects will see a lot of great possibilities and great potential. Let us then explore a few trends that seem to shape current automotive software engineering. In particular, let us explore the following trends:

- Autonomous driving—how the introduction of autonomous driving shapes the automotive sector and the software needed to steer cars.
- Self-*—how the ability to develop self-healing and self-adaptive systems influences the way in which we can design software in modern cars.
- Big data—how the ability to communicate and process large quantities of data changes the way we think about decision making in cars.
- New software development paradigms—how new software engineering methods influence the way we develop software for automotive systems.

In the remainder of this chapter we go through these trends.

M. Staron, *Automotive Software Architectures*,
DOI 10.1007/978-3-319-58610-6_9

9.2 Autonomous Driving

Undoubtedly the main trend in modern software in cars' is autonomous driving software. Autonomous driving software allows drivers to skip controlling the car or some of its functions. The NHSTA (National Highway Safety Traffic Administration) in the United States recognizes the following levels of autonomous functionality in cars [A+13]:

- Level 0, No automation—there are no functions in the car that can drive the car or support the driver.
- Level 1, Function-specific automation—according to the definition "automation at this level involves one or more specific control functions", meaning that certain functions can be autonomous, e.g. adaptive cruise control.
- Level 2, Combined function automation—where a group of functions can be automated and be autonomous. The driver, however, is still responsible for the control of the vehicle and must be prepared to take control of the vehicle on very short notice. Example functions are self-driving on highways.
- Level 3, Limited self-driving automation—the vehicle is able to drive autonomously under certain conditions and monitor the conditions; the drivers might need to occasionally take control, but the transition time is comfortably longer than at level 2.
- Level 4, Full self-driving automation—the vehicle is able to perform the entire trip autonomously; the driver is only expected to enter constraints and the destination for the trip. The level applies to both manned and unmanned vehicles.

One can see that modern vehicles already provide functions for automation Level 2 (combined function automation) and some even for level 3 (e.g. Tesla's autopilot functionality, [Pas14, Kes15]). This kind of functionality puts a lot of constraints on the automotive software.

First of all, this drives the complexity of software and therefore the cost of its development, verification, validation and certification. As the self-driving functionality is safety-critical it requires specific validation. It also requires complex reasoning in traffic situations on a very abstract level—e.g. whether it is better to save lives of the car's passengers or the lives of others in the accident.

Second of all, this kind of functionality drives the need for large quantities of data to process, which drives the need for processing power in modern cars. The processing power requires efficient CPUs and electronic buses of high throughput, which require more advanced infrastructure (e.g. cooling fans), that is often susceptible to environmental influences such as vibrations, humidity and temperature. This means that new components need to be develop especially for the cars, which drives costs.

Third of all, we need to understand that the quality of the sensors today is insufficient for advanced scenarios. Cameras are able to see clearly in specific conditions, but the human eye is still better synchronized with the human brain in all situations. Therefore cameras are not able to work effectively in low light or bad

weather conditions [KTI+05]. Using high-end cameras and sophisticated equipment would drive up the cost and still not guarantee the same quality as from human eyes and brains.

And finally, this kind of autonomous functionality requires acting on higher abstraction levels. Information about distance to the nearest obstacle needs to be transformed to a worldview which can be compared to a map view to determine the best course of action in a specific situation [BT16]. This requires more advanced algorithms which can be based on heuristics. The heuristics, however, are very challenging to prove to work correctly in all kinds of traffic situations, thus posing problems for safety certification.

9.3 Self-*

Self-healing is the ability of the system to autonomously change its structure so that its behaviour stays the same. An example concept of self-healing can be seen in the work of Keromytis et al. [Ker07], who define the self-healing as the ability to autonomously recover from erroneous execution.

One of the most prominent mechanisms used in self-healing systems is the MAPE-K (Measure, Analyse, Plan and Execute + Knowledge, [MNSKS05]). It is shown in Fig. 9.1 as an overwatch algorithm for an ECU realizing the adaptive cruise control functionality.

The algorithm in short is based on monitoring the execution of the algorithm for correctness. In the example of adaptive cruise control, we can monitor the radar to confirm it provides reliable results (e.g. no distortion is present). The analysis component checks whether one of the failure conditions has been detected (e.g. too

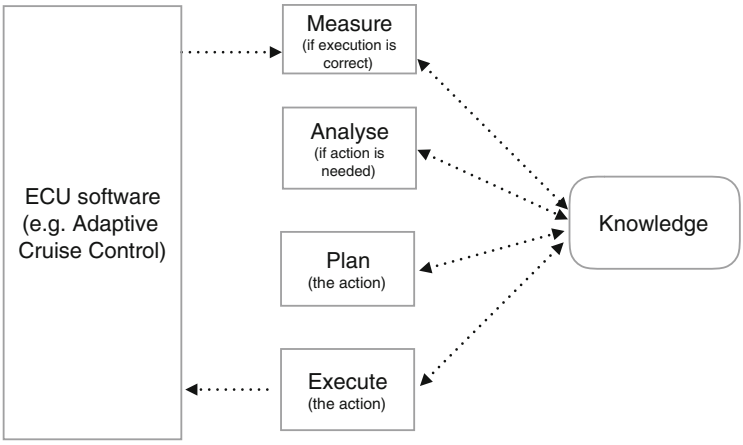

Fig. 9.1 Realization of MAPE-K for ECU software

much noise in the radar readings) and sends a signal to the plan component which plans appropriate action based on the reading and analysis. One of the actions can be to disable the adaptive cruise control and inform the user. Once the component makes a decision about the recovery strategy it moves to the execution and executes the repair strategy (i.e. informs the user and disables the adaptive cruise control algorithm).

This trend of using self-adaptation is used increasingly in safety-critical systems as it allows us to change the operation of a component in the presence of errors and failures. It can provide the ability to the system to self-degrade the functionality (e.g. temporarily change the operation of the engine, as discussed in Chap. 6).

However, there are still challenges which need to be addressed in order to make self-adaptation even more applicable to automotive systems. One of the major challenges is the ability to prove that the system is "safe" (in the sense of ISO/IEC 26262) during self-adaptation. Another is the fact that self-adaptation algorithms can be complex and need to be validated, but in many situations the failure modes cannot be replicated in real life. For example, it is difficult to safely replicate the situation where a radar in adaptive cruise control is broken when a vehicle drives at 150 km/h.

Nevertheless, we can perceive more self-* algorithms entering automotive systems as they need to monitor the increasingly complex decision algorithms in modern cars (e.g. related to autonomous driving).

9.4 Big Data

With the ability of modern cars to communicate with each other and the ability to use their own sensors in decision making, the amount to data used in modern cars has increased exponentially. At the same time, the field of computer science has evolved and started to tackle challenges related to storing, analysing and processing large quantities of data [MCB+11, MSC13].

Big Data systems are often characterized by the so-called five Vs:

- Volume—big data systems have large amounts of data (e.g. tera- or petabytes), which makes storage and processing a challenging task requiring new types of algorithms.
- Variety—the data comes from heterogeneous sources, has different formats, and has multiple semantic models, which require preprocessing before the data can be fed to analysis algorithms.
- Velocity—the data is provided at high speeds and requires processing realtime (e.g. from multiple sensors in the car and needs to be used to make safety-critical decisions). The speed requires large processing power, which might not be available in such systems as the automotive software.
- Value—the data collected has some business value (e.g. data about the driving routines of cars) which makes the storage, privacy and security issues

challenging, especially in combination with the velocity of processing and the next V—veracity.
- Veracity—the data has varying degree of quality, e.g., in terms of accuracy and trustworthiness. This varying degree of accuracy makes it challenging for the systems to use.

The challenges of using big data in automotive systems are related to all of the above V's. The large volume of data which comes from the car's own sensors needs to be processed and often stored, which puts requirements on storage in cars. Before the popularization of the SSD (Solid State Disk) technology it was rather challenging to use hard disks to store data (durability problems due to vibrations). Now, it is possible to store more data and also to process more data.

The high speed of processing requires more processing power, more efficient processors which take power and more connectivity. This drives the cost of automotive hardware since the more efficient processors require more infrastructure (stability, cooling), which is prone to problems in the automotive environment (humidity, vibrations). The hardware price is so important in the automotive domain (as opposed to other domains, where hardware is considered cheap) that one usually takes a calculation (a rule of thumb) that one dollar more expensive hardware per ECU can lead to 100 dollars more expensive cars.

The veracity of the data is a challenge as in many cases the "true" values cannot be measured but computed. For example, the slippage of the road in winter conditions cannot be measured but are derived either from ABS usage or the steering wheel friction. In some cases the data is obfuscated in order to secure privacy (e.g. triangulation algorithms to hide the true position of a car), which prevent the algorithms from "knowing" the true value of the data point [SS16].

In the future we will see more of big data, as large quantities of data are needed for autonomous driving and for advanced algorithms for collision prevention and avoidance.

9.5 New Software Development Paradigms

Software engineering for automotive systems has evolved the pace of the automotive domain. So, let us look into a few of the trends which shape the field today and will potentially shape the field in the future.

Agility in Specification Development Agile software development has been used in many domains outside of the automotive and now there is evidence that it is used increasingly in the automotive domain. In particular, at the lower part of the V-model suppliers work more agilely with their requirements engineering and software development [MS04]. We can also observe these trends scaling up to complete vehicle development [EHLB14] and [MMSB15]. With this increased adoption of Agile principles we can foresee the increased ability to specify requirements alongside software development, especially as the trends in automotive electronics

increasingly contain more commodity (or off-the-shelf) components. AUTOSAR also prescribes a standardized approach to development, which eases the use of iterative development principles as the development of electronics/hardware is decoupled from the development of functions/software.

Increased Focus on Traceability The increased amount of software in cars and their increased presence in safety systems leads to stricter processes for keeping track of requirements for safety-critical systems. ISO 26262 (Road vehicles—Functional Safety) is one example of this. In the automotive domain this means that the increased complexity of software modules [SRH15] leads to more fine-grained traceability management. One of the enablers of this increased traceability is the increased integration between the tools—tool chaining [BDT10] and [ABB+12].

Increased Focus on Non-functional Properties The increased use of software for active safety systems calls for increased focus on non-functional properties of software. The increased traffic on communication buses within the car, and the increased capacity of the communication buses call for more synchronization and verification. Safety analyses such as control path monitoring, safety bits and data complexity control, are just a few examples [Sin11]. As the focus of requirements engineering research in the automotive domain was mainly (or implicitly) in the functional requirements, we foresee an increased growth of research and emphasis on the non-functional requirements.

Increased Focus on Security Requirements A dedicated group of requirements is the security requirements, as our cars are increasingly connected and therefore prone to hacker attacks [SLS+13] and [Wri11]. The recent demonstration of the possibility of steering a Jeep Wrangler vehicle offroad showed that the threat is real and related to the safety of cars and transport systems. We therefore perceive that the ability to prevent attacks will the focus of the automotive software development increasingly more in the coming decade.

9.5.1 Architecting in the Age of Agile Software Development

Architecture development in software development is usually conducted by experienced architects, and the larger the product, the more the experience required. As each type of system has its specific requirements, the architectural design requires attention to specific aspects like realtime properties or extensibility. For example, in the telecom domain the extensibility and performance are the main aspects, whereas in the automotive domain it is safety and performance that are of the utmost priority. The architecture development efforts are dependent to some extent on the software development process adopted by the company, e.g. the architecture development methods differ in the V-model and Agile methodologies. In the V-model the architecture work is mostly prescriptive and centralized around the

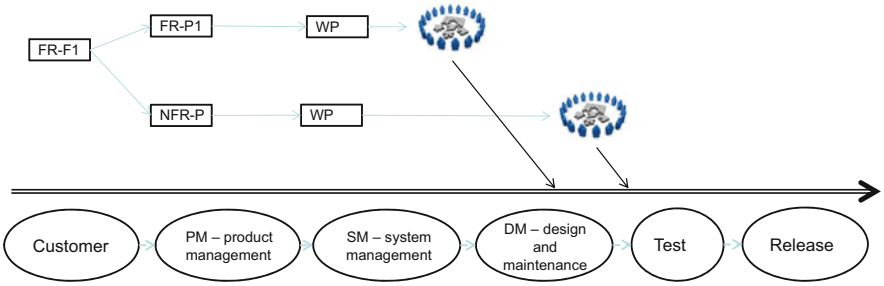

Fig. 9.2 Feature development in Lean/Agile methods

architects whereas in the Agile methods the work can be more descriptive and distributed into multiple self-organized teams.

As Agile software development principles spread in industry, architecture development evolved. As Agile development teams became self-organized, architecture work became more distributed and harder to control centrally [Ric11]. The difficulties stem from the fact that Agile teams value independence and creativity [SBB+09] whereas architecture development requires stability, control, transparency and proactivity [PW92]. Figure 9.2 presents an overview of how the functional requirements (FR) and non-functional requirements (NFR) are packaged into work packages and developed as features by teams. Each team delivers code to the main branch. Each team has the possibility to deliver the code to any component of the product.

The requirements come from the customers and are prioritized and packaged into features by product management (PM), which communicates with system management (SM) on the technical aspects of how the features affect the architecture of the product. System management communicates with the teams (DM, Test) that design, implement and test (functional testing) the feature before delivering it to the main branch. The code in the main branch is tested thoroughly by dedicated test units before being release [SM11].

9.6 Other Trends

Bosch has presented three trends which shaped software engineering in the mid-2010s [Bos16]: speed of software development, ecosystems and data-driven development. He predicted that the companies which are the first ones on the market would be more successful than others as the innovation model is based on the shark's tail rather than the traditional technology adoption curve. In particular, the majority of new, innovative software products are adopted by the market at a tremendous pace, and then companies need to be prepared to be ready for the market. Followers do not have the same ability to attract customers [DN14]. Ecosystem thinking (e.g.

Apple's App store or Google's Play store) has been present in the automotive sector from way back in the hardware domain (e.g. customers of BMW are bound to buy spare parts from manufacturer) but not in the software domain. And finally we have data-driven development and the Lean innovation thinking [Rie11] where customers provide the companies with the data on how to develop their products. With connected cars and the ability to update the car software over the air we will probably see more data-driven development in the automotive industry in the coming decade.

Burton and Willis from Gartner identified five mega-trends which have the potential of shaping software engineering in the coming decades [BW15]. These mega-trends are:

- Digital Business Moves Toward the Peak of Inflated Expectations
- IoT, Mobility and Smart Machines Rapidly Approach the Peak
- Digital Marketing and Digital Workplace Quickly Move Up
- Analytics Are at the Peak
- Big Data and Cloud Make Big Moves Toward the Trough of Disillusionment

In short, these trends will drive the need for more advanced functionality of cars and the use of big data for decision making and even the development of the cars (finding out the requirements from the data rather than focus group interviews). However, they predict that the era of wearables (e.g. smartwatches) will reach the so-called "pit of disillusion" where they will probably reach the state where no more development is of interest to the customers.

In their 2016 report, Gartner Associates provide even more focus on Artificial Intelligence, Machine Learning and autonomy. We perceive these technologies as new hype in automotive software engineering, especially when combined with different levels of autonomy and self-adaptation algorithms. This will mean even more complexity and software in future cars.

9.7 Summary

To conclude this chapter let us make a speculation that future cars will be more like computer platforms where different third party companies can build applications. We can see the self-driving car of Google as an example of such a move [Gom16].

The telecommunication domain has evolved from proprietary solutions in mobile phones of the 1990s to standardized platforms and ecosystems of the smartphones of the 2010s—Android and iOS leading the field in this direction. Customers buying a new mobile phone buy a device which they can load with apps of their own choice—some free and some paid. We can see that the ability to update car's software will lead to similar trends (already visible in the infotainment domain.)

These possibilities of opening up for third party software in cars is expected to change the face of the automotive industry in the future. Commoditizing platforms and portability between vendors on the application level can cause cars to become

much safer and much more fun. We can expect the cars to become hubs for all kinds of devices and integrated with wearables to provide drivers and passengers with an even better driving experience than today's. We need to live and see what the future of software in cars will bring.

References

A+13. National Highway Traffic Safety Administration et al. Preliminary statement of policy concerning automated vehicles. *Washington, DC*, pages 1–14, 2013.

ABB+12. Eric Armengaud, Matthias Biehl, Quentin Bourrouilh, Michael Breunig, Stefan Farfeleder, Christian Hein, Markus Oertel, Alfred Wallner, and Markus Zoier. Integrated tool chain for improving traceability during the development of automotive systems. In *Proceedings of the 2012 Embedded Real Time Software and Systems Conference*, 2012.

BDT10. Matthias Biehl, Chen DeJiu, and Martin Törngren. Integrating safety analysis into the model-based development toolchain of automotive embedded systems. In *ACM Sigplan Notices*, volume 45, pages 125–132. ACM, 2010.

Bos16. Jan Bosch. Speed, data, and ecosystems: The future of software engineering. *IEEE Software*, 33(1):82–88, 2016.

BT16. Sagar Behere and Martin Törngren. A functional reference architecture for autonomous driving. *Information and Software Technology*, 73:136–150, 2016.

BW15. Betsy Burton and David A Willis. Gartner's Hype Cycles for 2015: Five Megatrends Shift the Computing Landscape. *Recuperado de:* https://www.gartner.com/doc/3111522/gartners--hype--cycles--megatrends--shift, 2015.

DN14. Larry Downes and Paul Nunes. *Big Bang Disruption: Strategy in the Age of Devastating Innovation*. Penguin, 2014.

EHLB14. Ulf Eliasson, Rogardt Heldal, Jonn Lantz, and Christian Berger. Agile model-driven engineering in mechatronic systems-an industrial case study. In *Model-Driven Engineering Languages and Systems*, pages 433–449. Springer, 2014.

Gom16. Lee Gomes. When will Google's self-driving car really be ready? It depends on where you live and what you mean by "ready" [News]. *IEEE Spectrum*, 53(5):13–14, 2016.

Ker07. Angelos D Keromytis. Characterizing self-healing software systems. In *Proceedings of the 4th international conference on mathematical methods, models and architectures for computer networks security (MMM-ACNS)*, 2007.

Kes15. Aaron M Kessler. Elon Musk Says Self-Driving Tesla Cars Will Be in the US by Summer. *The New York Times*, page B1, 2015.

KTI+05. Hiroyuki Kurihata, Tomokazu Takahashi, Ichiro Ide, Yoshito Mekada, Hiroshi Murase, Yukimasa Tamatsu, and Takayuki Miyahara. Rainy weather recognition from in-vehicle camera images for driver assistance. In *IEEE Proceedings. Intelligent Vehicles Symposium, 2005.*, pages 205–210. IEEE, 2005.

MCB+11. James Manyika, Michael Chui, Brad Brown, Jacques Bughin, Richard Dobbs, Charles Roxburgh, and Angela H Byers. Big data: The next frontier for innovation, competition, and productivity. 2011.

MMSB15. Mahshad M Mahally, Miroslaw Staron, and Jan Bosch. Barriers and enablers for shortening software development lead-time in mechatronics organizations: A case study. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, pages 1006–1009. ACM, 2015.

MNSKS05. Manoel, E., Nielson, M.J., Salahshour, A., KVL, S.S. and Sudarshanan, S., 2005. *Problem determination using self-managing autonomic technology*. IBM International Technical Support Organization.

MS04. Peter Manhart and Kurt Schneider. Breaking the ice for agile development of embedded software: An industry experience report. In *Proceedings of the 26th international Conference on Software Engineering*, pages 378–386. IEEE Computer Society, 2004.

MSC13. Viktor Mayer-Schönberger and Kenneth Cukier. *Big data: A revolution that will transform how we live, work, and think*. Houghton Mifflin Harcourt, 2013.

Pas14. A Pasztor. Tesla unveils all-wheel-drive, autopilot for electric cars. *The Wall Street Journal*, 2014.

PW92. Dewayne E Perry and Alexander L Wolf. Foundations for the study of software architecture. *ACM SIGSOFT Software Engineering Notes*, 17(4):40–52, 1992.

Ric11. Eric Richardson. What an agile architect can learn from a hurricane meteorologist. *IEEE software*, 28(6):9–12, 2011.

Rie11. Eric Ries. *The lean startup: How today's entrepreneurs use continuous innovation to create radically successful businesses*. Random House LLC, 2011.

SBB+09. Helen Sharp, Nathan Baddoo, Sarah Beecham, Tracy Hall, and Hugh Robinson. Models of motivation in software engineering. *Information and Software Technology*, 51(1):219–233, 2009.

Sin11. Purnendu Sinha. Architectural design and reliability analysis of a fail-operational brake-by-wire system from iso 26262 perspectives. *Reliability Engineering & System Safety*, 96(10):1349–1359, 2011.

SLS+13. Florian Sagstetter, Martin Lukasiewycz, Sebastian Steinhorst, Marko Wolf, Alexandre Bouard, William R Harris, Somesh Jha, Thomas Peyrin, Axel Poschmann, and Samarjit Chakraborty. Security challenges in automotive hardware/software architecture design. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 458–463. EDA Consortium, 2013.

SM11. Miroslaw Staron and Wilhelm Meding. Monitoring Bottlenecks in Agile and Lean Software Development Projects–A Method and Its Industrial Use. *Product-Focused Software Process Improvement*, pages 3–16, 2011.

SRH15. Miroslaw Staron, Rakesh Rana, and Jörgen Hansson. Influence of software complexity on iso/iec 26262 software verification requirements. 2015.

SS16. Miroslaw Staron and Riccardo Scandariato. Data veracity in intelligent transportation systems: the slippery road warning scenario. In *Intelligent Vehicles Symposium*, 2016.

Wri11. Alex Wright. Hacking cars. *Communications of the ACM*, 54(11):18–19, 2011.