Continuous Integration Beyond the Team: A Tooling Perspective on Challenges in the Automotive Industry

Continuous Integration Beyond the Team: A Tooling Perspective on Challenges in the Automotive Industry[*]

Eric Knauss[1], Patrizio Pelliccione[1], Rogardt Heldal[1,2], Magnus Ågren[1], Sofia Hellman[3], Daniel Maniette[4]
[1]Chalmers University of Technology | University of Gothenburg
[2]Bergen University College
[3]ÅF Technology Embedded West
[4] Cybercom Group
{eric.knauss,patrizio.pelliccione}@cse.gu.se, {heldal,magnus.agren}@chalmers.se, sofia.hellman@afconsult.com, daniel.maniette@cybercom.com

ABSTRACT

The practice of Continuous Integration (CI) has a big impact on how software is developed today. Shortening integration and feedback cycles promises to increase software quality, feature throughput, and customer satisfaction. Thus, it is not a surprise that companies try to embrace CI in domains where it is rather difficult to implement.

In this paper we present our findings from two rounds of interviews with a car manufacturer on the use of tools in system engineering and how these tools would support wider adoption of CI. Our findings suggest a complex tool landscape with immense requirements that are not easily fulfilled by existing tools; this holds also for tools that well support CI in other domains. From this notion, we further explore what makes the automotive domain challenging when it comes to CI (namely complexity of system and value chain). We hope that our findings will help address such challenges.

Keywords

continuous integration, automotive systems engineering

1. INTRODUCTION

Continuous Integration (CI) is a practice that promises to shorten integration and feedback cycles [9, 19]. CI is reported to increase software quality and speedup feature development [19]. However, many implementations of CI have been on a team level, where this practice works exceptionally well [9, 17]. In order to fully leverage the potential advantages, CI should be implemented organization-wide [16, 17] or even lead to continuous delivery of value to the customer [15, 14].

Many categorizations of software development domains exist. We found the categorization in *services*, *enterprise*, and *product* domains by Bass particularly helpful [3]. Accordingly, a company providing software-based services online will have the easiest task of implement CI, whereas companies that develop software embedded into a physical product will struggle more with CI adoption (in the scope of this paper we omit discussion of enterprise software). This notion is supported by our literature review: while good experiences with these practices have been reported in some domains [9, 5], there has been also reports on challenges with implementing these practices especially in embedded and automotive domains [6].

Today, the automotive business is rapidly changing, for example because of to the appearance of new major players in the field (Google [10], Apple [1, 2], UBER and others need to be seen as competitors to more traditional automotive companies). Thus, automotive companies have a strong motivation to embrace organization-wide CI to yield improvements in flexibility and cycle time, despite the challenges.

In this paper, we rely on two batches of interviews on tooling (i) for system engineering and (ii) for CI to explore the underlying challenges and impediments for CI in the automotive domain. Through the understanding of requirements for systems engineering tooling that supports these practices, we gain an understanding of the underlying challenges that make it so difficult to introduce organization-wide – or at least system-wide – implementations of them. Particularly, we explore the following research questions:

RQ1: How well do tools used at the moment support CI in the automotive domain?

RQ2: What new needs towards tooling exist in the automotive industry?

Our key findings include that tooling that did work well in a V-model context is causing friction in a continuous engineering environment, especially because it encourages silo thinking and does not support cross-functional aspects. In addition, existing tools that do work well in other domains cannot be easily adapted to automotive systems engineering. From these findings for RQ1 and RQ2, we derive specific impediments around the complexity of automotive systems engineering and complexity of value chains.

[*]This work was funded by the Vinnova FFI projects Next Generation Electronic Architecture and Next Generation Electronic Architecture step 2.

ESEM '16, September 08-09, 2016, Ciudad Real, Spain
ACM 978-1-4503-4427-2/16/09.
http://dx.doi.org/10.1145/2961111.2962639

Some of these challenges are specific to automotive, others might very well apply to any product company embracing CI. The paper is organized as follows. Section 2 describes the complexity of the automotive domain. Section 3 describes the research method, Section 4 presents and discusses our findings. Finally, the paper concludes with final remarks and directions for future research in Section 6.

2. CONTEXT

The automotive domain is extremely interesting to investigate at this point in history, since automotive companies are encumbered with the development of electrical, autonomously driving, and connected cars. Moreover, new players, like Google, Apple, and Tesla, are entering the automotive domain, and this is influencing the way and speed of producing cars. Historically, software was introduced in cars to optimize the control of the engines. Since then the growth of software within the car has been exponential for each year and today not a single function is performed without the involvement of software. According to industry experts, 80% to 90% of the innovation within the automotive industry is based on electronics, as mentioned for instance by Peter van Staa - Vice-President Engineering of Robert Bosch GmbH at the European Technology Congress in June 2014[1]. A big part of electronics is software. Moreover, the most advanced cars have more (or at least a comparable amount of) software than fighter airplanes[2]. Vehicles are also produced in higher volumes than airplanes, which leads to harder requirements on the technology cost.

A modern car is driven by a network of often more than 100 small computers (ECUs - Electrical Control Units), running thousands of (software) components to fulfil user visible functions and to control sensors and actuators in a distributed fashion. The high level of connectedness throughout the car network characterizes the complexity of automotive engineering and it is interesting to investigate how such an organization can adopt CI.

Organization challenges: The complexity of the automotive domain leads to having different abstraction levels, where some organizational sections take care of top level requirements, other sections produce the architecture, others yet the design, and still others develop the software, hardware and mechanical parts for the product. Even on the development level there are many components that can be done by only a few specialized teams, such as controlling the engine, or making the software for the braking system. Thus, specialized teams exist both on the same abstraction level, as well as on different abstraction levels. As we will see in the result all these different organizational sections have an impact when it comes to tools and toolchains.

Moreover, it is very impractical to produce all parts comprising a vehicle in-house. For this reason OEMs normally ask suppliers to produce parts of the vehicle, and the number of suppliers involved in parallel development can easily reach a count of 50 different suppliers. That means that the knowledge and competence of building a car is divided among many companies. Consequently, there can be very many companies with different organizations, objectives, competences, constraints, cultures, languages, and geographical placements, involved in a car project. The agreements between these companies are currently mainly based on contracts. Contracts are a killer for Agile development.

[1]http://www.etcwroclaw.eu/files/presentations/peter_van_staa.pdf

[2]As said by Alfred Katzenbach, the director of information technology management at Daimler: http://spectrum.ieee.org/transportation/systems/this-car-runs-on-code

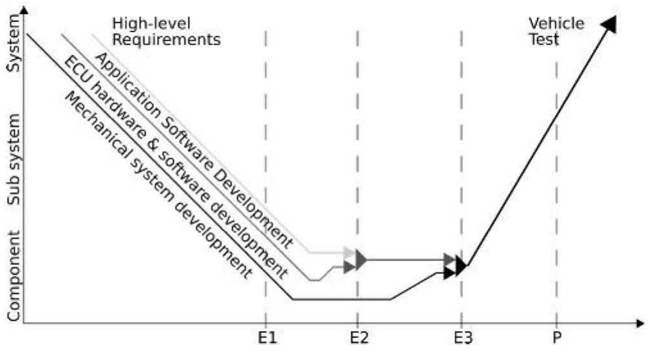
System
Sub system
Component
High-level Requirements
Application Software Development
ECU hardware & software development
Mechanical system development
Vehicle Test
E1
E2
E3
P

Figure 1: V-model used within Automotive industry

Different speeds within the same company: Mechatronic development, such as for a modern car, involves software, hardware and mechanical development. The widely adopted model for this distributed development is the V-model, see Figure 1. The V-shape represents the journey from requirements to a complete car.

As shown in Figure 1, the overall system development process at automotive companies is a sequential process with a number of points in time where artifacts need to be delivered. There are three tracks of parallel development: (a) the software components (SWC), (b) the hardware like ECU, and (c) mechanical parts. E1-E3 are electronic integration points and P is where the software should be production ready. This is probably one of the main differences between *service* and *product* companies: in a service company, as soon as some software is ready it can be deployed, since hardware and mechanical parts are not part of the picture. In contrast, product companies are forced to develop new ways of working to reach agile development and CI. We will discuss this both in the results section, i.e. Section 4 and in the discussion section, i.e. Section 5.

Challenges coming from variability and safety: Typically, several cars are built on the same platform according to a product-line engineering approach; this inevitably creates a lot of variation and a lot of variants. In addition, customers can choose from a multitude of variants and regulations differ between countries, leading to even more variation. Consequently, the distributed software within a car can be configured in a million ways. This requires some upfront work: without a good architecture it is not possible to handle the variants. There are papers claiming that it would be hard to succeed without product line development [4]. Interestingly, even though agile development and Software Product Line Engineering (SPLE) have similar claims for what concerns productivity, these approaches are very different; in fact SPLE involves significant upfront commitment and a disciplined approach [13].

Safety is an unavoidable constraint of the automotive industry. The ISO26262 standard [11] influences organization structure, development process, adopted tools, artifacts that should be produced and released, etc. Such safety concerns require quite a bit of upfront work and limit agility.

3. METHOD

To answer the research questions we used an exploratory case study design. We collected qualitative data from semi-structured interviews within one *Automotive OEM*. We had in total 26 interviews in two related, but separate scopes.

In the first scope, 10 interviews mainly focused on tool aspects and specifically on how tools are used for system engineering, as well as challenges the company is encountering when moving towards company wide CI. We interviewed practitioners from four different departments in the organization, namely: *Electrical Systems Architecture*, *Infotainment*, *Powertrain*, and *Body Electronic Software*. We selected these sections since they have responsibilities at different abstraction levels in the V-model.

In the second scope, 16 interviews focused on CI in software developing teams within the company. We recruited interviewees from 8 departments (including the ones covered in the first scope) and talked to two representatives from each department. Each interview lasted around one hour.

In both scopes, we validated the collected data by sending notes taken during the interviews back to the interviewees, and by having the case company do a final, thorough, review of the compiled notes. We then discussed common observations and extracted themes through several workshops between the authors.

4. RESULTS

An important factor of CI is that it requires automation, especially when it comes to building and integrating. Such automation is useful for the automotive industry as well, yet as discussed in Section 2 it is difficult to achieve. Challenges include that flexibility and speed of software development need to be balanced with production and maintenance cost of long-living, physical products. In addition, building an automotive system requires contributions from many disciplines (such as mechanics and mechatronics) as well as a large number of suppliers. Thus, how to design a change and how to adjust the architecture of the system is an important consideration of CI beyond the team-level in automotive. It is an open, but relevant question of how to support agile ways of working, i.e. short cycle times based on small changes of the system in order to satisfy a customer request as well as the ability to apply those changes to a live system.

4.1 RQ1: How Well Do Tools Used at the Moment Support CI in the Auto. Domain?

The answers from our interviewees with respect to this research question can be categorized in two themes: *(Theme 1.1) impact from organizational specialization on tools*, and *(Theme 1.2) tools themselves as show stoppers for increased agility.*

4.1.1 Theme 1.1: Impact from Organizational Specialization on Tools

In order to allow a large organization to function, shared tooling platforms need to be provided. For this reason, the *Automotive OEM* made a decision to use *System engineering tool 1*[3] for managing system engineering data, which is a domain specific, internal tool. However, the ways of working are highly optimized and specialized in the different departments and thus require different capabilities of tools. This has lead to the use of different tools for similar purposes in the different departments which constantly strive for increased excellence. For example, *Body Electronic Software*, a department that develops software in-house, has special needs and requests towards version control and branching, handling of component interfaces, and backwards compatibility. In *Electrical Systems Architecture* tools such as *Architecture model tool* are used, see Figure 2; the focus is on tools providing visualizations. Figure 2 gives an overview of the different tools we encountered in our study.

During our interviews, we found it striking that different tools were used for similar or at least comparable tasks. Also, tools like *Architecture model tool* are successfully used in other domains to cover similar use cases as *System engineering tool 1*, i.e. modeling the architecture of the car and defining the system design. *Architecture model tool*is highly adaptable, e.g. through defining profiles and does allow defining the system design in sophisticated graphical models. In contrast, *System engineering tool 1* does not provide graphical models. This allows this tool to scale for the whole organization and to cover the whole system under development in sufficient detail to support CI organization wide. This makes it a great tool for managing and disseminating design and architecture, despite its limitations of graphical modeling. Thus, we found that tools like *Architecture model tool*are used for drafting the architecture within the local scope of a department. Such architectural models often express the planned architecture. In contrast, *System engineering tool 1* is the reference for every implementation activities in CI and accurately represents the current architecture. Even if a more homogeneous tool landscape should be established, it would still be necessary to allow parallel work on both planned and current architecture to support CI beyound the team.

4.1.2 Theme 1.2: Tools Themselves as Show Stoppers for Increased Agility

The previous theme covered *Architecture model tool* and *System engineering tool 1*. These tools are very complex, and hard to learn and use. Worse, they have more functionality than needed in many cases. Still, neither of them are satisfactory for the (whole) job, even when combined. Dealing with variants requires involving other tools; dealing with safety requires involving yet further tools. The *Automotive OEM* that we have interviewed has a product life-cycle management tool – *PLM tool* – which keeps the specifications produced by the different tools in one place.

The use of different tools in this way adds an extra layer of complexity, as can be seen in Figure 2, which gives an overview of the complex distribution of tools over the development process. The figure depicts that the electrical systems architecture is not really included in the rest of the development part of the V-model. Parts of the electrical architecture, such as the logical topology, are included in *System engineering tool 1*, but the UML models are neither in *System engineering tool 1* nor in *PLM tool*. Information transfer between *PLM tool* and *Architecture model tool* is handled as knowledge transfer between individuals, and where there is data transfer between tools, it is often manual.

[3]In this study we generalize from concrete tool names, since we are interested in the underlying needs when adopting CI, not in the tools themselves.

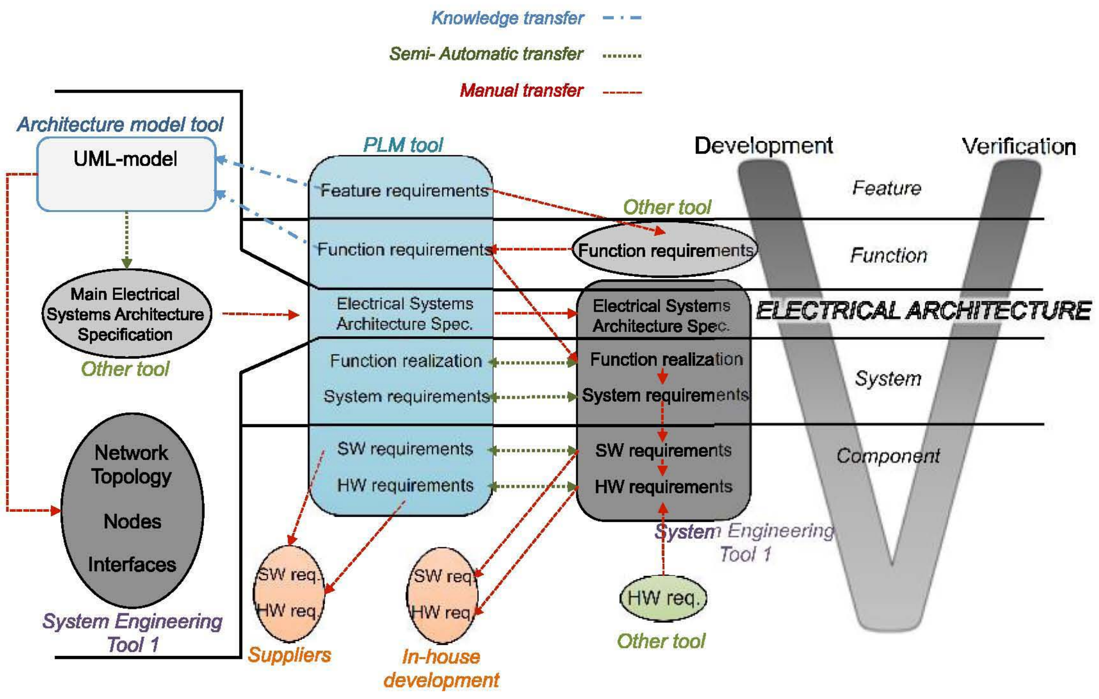

Figure 2: Current tool usage over the development process.

For development done by external suppliers, requirements on hardware and software are manually extracted from *PLM tool*. For development done in-house, similarly purposed requirements are extracted, still manually, from *System engineering tool 1*. This points to the need expressed by the interviewees for connections between tools, for example by open APIs, and for coherence in the toolchain.

System engineering tool 1 supports producing boxes as placeholders for code. Input and output signals of the boxes can be seen and how the boxes fit together can be studied; it is not possible to run anything, though, since the boxes do not contain any behavior. This could be a hindrance to agility, but also an opportunity, since the resulting parts to be implemented are isolated.

Using many different tools is fine as long as they can be plugged together. Currently, though, the tool vendors often offers single tools covering the whole development process, leading to multiple overlapping tools with different strengths. Having one tool that fits all the needs might be a solution, however, it is not a likely scenario and also comes with the problem of vendor lock-in. What is needed is better ways of combining tools. It is not only a tool problem, though: different sections within the company have to find better ways of working. For example, in order to support continuous integration, *Electrical Systems Architecture* needs to more strongly influence the development and when changes are needed by development, architects should also be involved.

4.2 RQ2: What New Needs Towards Tooling Exist in the Automotive Industry?

The answers from our interviewees with respect to this research question can be categorized in four themes: *(Theme 2.1) one tool does not fit all needs*, *(Theme 2.2) scalability by simulation and visualization*, *(Theme 2.3) fit tools to processes, not the other way around*, and *(Theme 2.4) don't always blame the tools.*

4.2.1 Theme 2.1: One Tool Does Not Fit All Needs

With the tools used and the work procedures differing between sections of the organization, it may be difficult to have only one tool environment covering the entire electrical systems development and verification. Many of the tool suppliers studied in this work aim to cover the entire V-model with their tools, but parts are still missing. This, together with the different needs of the sections within the company confirms the need for section specific tools.

The views on having a single tool environment differs between the interviewed sections. Whilst all seem to agree on the need of the same line of thought through the toolchain, *Electrical Systems Architecture* argues that as long as the tools have open APIs and well documented file formats there need not be interfaces between the tools, whereas another section argues that the tools need to have good interfaces between each other.

4.2.2 Theme 2.2: Scalability by Simulation and Visualization

One main request has been mentioned in all interviews: the possibility to use model based development instead of document or text based requirements. This includes having model based requirements that can be used through the entire toolchain, being able to model the system at different levels, and having the models at different levels connected to each other. Consequently, the models and modeling tools have to be scalable as well.

Executable models, such as Simulink, play an important role in the automotive industry, since they are used for software development, but also for simulation of hardware and mechanical parts. Therefore, executable models need to be continuously integrated and tested. Simulation and virtual verification promises to offer a scalable and efficient solution in such a model-driven systems engineering environment.

One problem today is that the tools do not always handle enough simultaneous users for this high level of interconnectedness. For example, modeling tools at the *Automotive OEM* need to support more than 250 simultaneous users, limiting the choice of suitable tools significantly. We argue that this huge amount of simultanuous users not only shows the huge complexity of automotive systems engineering, but also is a crucial enabler for CI beyond the team level by providing real-time information about the current architecture (which is constantly evolving through continuously integrated changes). We discuss modeling further in Section 5.

In order to support continuous integration over all abstraction levels, *Electrical Systems Architecture* has the need for a tool where it is possible to get an overview of the electrical system without seeing all the details in *System engineering tool 1*, to avoid information overload. There is a need for connecting the planned architecture and current electrical architecture [8]. One way forward is to have tools to visualize the architecture produced with *System engineering tool 1* in a suitable level of abstraction to allow for architectural decisions on the complete system under development.

4.2.3 *Theme 2.3: Fit Tools to Processes, Not the Other Way Around*

CI is an advanced, flexible way of working that, if scaled beyond individual teams, promises to accelerate collaboration between organizations. It is important that the toolchain supports this, but we found that at least in the complex domain of automotive systems engineering further research about how to fit tools better to CI in complex product organizations is needed.

Due to the lack of suitable tooling, in-house development seems to be a natural strategy. However, the *Automotive OEM* does not wish to become widely involved in tool building. Sometimes this cannot be avoided and we found several domain specific tools that were developed in-house to overcome a lack of existing tools. An alternative strategy is having a strong collaboration between the company and the tool vendor, as is the case for the large tool *System engineering tool 1* within the *Automotive OEM*.

4.2.4 *Theme 2.4: Don't Always Blame the Tools*

The interviewees have indicated that many of the tools are complex to use. To do complex things, however, the tools themselves often need to be complex; not all problems can be blamed on the tools – the support around the tools is equally important. Our paper [20] discusses this issue in detail. We can find one concrete example in *System engineering tool 1* where agility is stopped by the process, not by the tool: For the network in the car, the specification of sendable signals can only be changed 4 times a year, a restriction imposed by the process, not by the tool.

5. DISCUSSION

In this section, we discuss some of the aspects that will play an important role for having CI in the automotive industry. Specifically: in Section 5.1 we discuss the importance of managing data across a toolchain; in Section 5.2 we discuss Model Driven Engineering (MDE) as an enabler for early feedback and integration; and finally, in Section 5.3 we investigate how the relationship between OEM and suppliers should change.

5.1 Flexible Data Management Toolchain

As outlined above products become more complex and developed by larger and distributed teams. This calls for more unified, controlled, and consistent data. Shahrokni et al. [18] discuss a new paradigm called organic evolution of development organizations. In this paradigm, the data management toolchain is conceived to be configurable and flexible so to enable an agile way of working and structuring on low and detailed levels of the process, while providing control and progress tracking on managerial levels.

5.2 MDE as an Enabler for Early Feedback

To be able to obtain quicker feedback in the automotive industry, Model Driven Engineering (MDE) has become popular [12, 7]. Executable models, in the form of Simulink models, are utilized to introduce feedback early in the development process. To be able to run these models at an early stage there is a need for models of hardware, software, and mechanical parts, as raised in our interviews, and shown in Figure 1. This enables different kinds of simulations, typically refereed as MIL (model in the loop), SIL (software in the loop), and HIL (hardware in the loop), corresponding to which parts of the mechatronic system that are physically available, and which are instead represented as models. In the long run, complete virtual cars can be considered.

This will give the opportunity to perform different types of verification and validation with the car immersed in virtually generated environments. That will then enable for instance the verification and validation of the behavior of the car in a system of systems setting, i.e. the car interacting with other cars, with pedestrians, cyclists and with a smart city. It will also facilitate early verification and validation of software updates within the virtual car. It will, however, demand good techniques and tools. Whether current tools can provide what they promise in form of simulations remains to be seen.

The use of models opens another interesting opportunity: exploiting them to facilitate collaboration between the OEM and the suppliers. Models representing mechanical and hardware parts might be exploited to test early the integration between software developed concurrently by different companies and/or organizations. CI could thus be possible even between separate companies. An additional related aspect is to have a tight connection between models and code so to effectively exploit the confidence gained on the virtual vehicle. Promising techniques for enabling this tight connection are model-to-model or model-to-code transformations.

5.3 A New Way of Working With Suppliers

CI calls for a new way of working between OEM and its suppliers. The relation between the OEM and its suppliers should change from a contract-based relationship to a more collaborative one where the OEM and suppliers are part of the same ecosystem and collaborate for mutual benefit. More transparency and less contract-based working will then be some of the key enablers for CI.

6. CONCLUSIONS AND FUTURE WORK

In this study, we aimed at investigating from a tool perspective which impediments exist for implementing CI in the automotive industry. By investigating how current tooling supports adoption of organization-wide CI, we have a unique opportunity to understand how the needs of a car manufacturer change during adoption of CI. Our main finding here is that existing tooling is the result of old specializations and silos (see Themes 1.1 and 1.2). In order to allow quicker cycle times, these silos need to be overcome to support cross-functional collaboration (see Theme 2.3). This will entail major changes in the current toolchain. Secondly, we investigated which new needs towards tooling exist and we found scalability to be a driving force here (see Theme 2.2). Models and model-driven engineering are important for automotive software development, and in CI such models need to be integrated on the system level, thus bringing together knowledge about application software, ECU hardware/basic software, and mechanics.

Based on these findings, we suggest that a new way of working needs to also investigate new business models. Knowledge about the different domains is distributed over several organizations in the automotive value chain, including suppliers and consultants. In order to facilitate CI, ways have to be found to quickly create a realistic environment where changes can be integrated. Construction of shared plant models in a particular value chain as a target for CI can be valuable future research. Further, in order to address the need for scalability and cross-functional collaboration, we anticipate the need to establish suitable transparency between the actors in the automotive value chain, to avoid bottlenecks as well as information overload.

7. REFERENCES

[1] Apple gears up to challenge tesla in electric cars. *Wall Street Journal*, Feb 13, 2015. Last visit: 2015-March-10.

[2] Apple hiring automotive experts to work in secret research lab. *Financial Times*, Feb 14, 2015. Last visit: 2015-March-10.

[3] M. Bass. Software engineering education in the new world: What needs to change? In *2016 IEEE 29th Conf. on Software Engineering Education and Training*. IEEE, 2016.

[4] L. Brownsword and P. Clements. A Case Study in Successful Product Line Development. Standard CMU/SEI-96-TR-016,ESC-TR-96-016, 1996.

[5] A. Debbiche, M. Diener, and R. B. Svensson. Challenges when adopting continuous integration: A case study. In A. J. et al., editor, *Proc. of the 15th Int. Conf. of Product Focused Software Development and Process Improvement (Profes)*, volume 8892 of *LNCS*, pages 17–32, Helsinki, Finland, 2014. Springer.

[6] U. Eklund and J. Bosch. Applying agile development in mass-produced embedded systems. In C. Wohlin, editor, *Proc. of Int'l Conf. on Agile Softw. Dev. (XP)*, volume 111 of *LNBIP*, pages 31–46, Malmö, Sweden, 2012. Springer.

[7] U. Eliasson, R. Heldal, J. Lantz, and C. Berger. Agile model-driven engineering in mechatronic systems - an industrial case study. In *Model-Driven Engineering Languages and Systems - 17th Int. Conf., MODELS 2014, Valencia, Spain, September 28 - October 3, 2014. Proceedings*, pages 433–449, 2014.

[8] U. Eliasson, R. Heldal, P. Pelliccione, and J. Lantz. Architecting in the automotive domain: Descriptive vs prescriptive architecture. In *Software Architecture (WICSA), 2015 12th Working IEEE/IFIP Conf. on*, pages 115–118. IEEE, 2015.

[9] M. Fowler. Continuous integration. Technical report, 2006. http://martinfowler.com/articles/continuousIntegration.html last visit: 2016-01-12.

[10] E. Guizzo. How the google self-driving car works. *IEEE Spectrum*, Oct 18, 2015. Last visit: 2015-March-10.

[11] Road vehicles – Functional safety. Standard ISO 26262, 2011.

[12] J. Lantz. Using models to scale agile mechatronics development in cars: case studies at volvo car group. In *18th Int. Software Product Line Conf., SPLC '14, Florence, Italy, September 15-19, 2014*, page 20, 2014.

[13] K. Mohan, B. Ramesh, and V. Sugumaran. Integrating software product line engineering and agile development. *IEEE Software*, 27(3):48–55, 2010.

[14] S. Neely and S. Stolt. Continuous Delivery? Easy! Just Change Everything (well, maybe it is not that easy). In *Proc. of Agile Conf.*, pages 121–128, Nashville TN, USA, 2013. IEEE.

[15] O. Rissanen and J. Münch. Transitioning Towards Continuous Delivery in the B2B Domain: A Case Study. In C. L. et al., editor, *Proc. of 16th Int. Conf. on Agile Processes in Software Engineering and Extreme Programming (XP)*, volume 212 of *LNBIP*, pages 154–165, Helsinki, Finland, 2015. Springer.

[16] M. Roberts. Enterprise continuous integration using binary dependencies. In *Extreme Programming and Agile Processes in Software Engineering*, pages 194–201. Springer, 2004.

[17] R. Rogers. Scaling continuous integration. In *Extreme Programming and Agile Processes in Software Engineering*, pages 68–76. Springer, 2004.

[18] A. Shahrokni, P. Gergely, J. Söderberg, and P. Pelliccione. Organic evolution of development organizations - an experience report. In *SAE 2016 World Congress and Exhibition - Model-Based Controls and Software Development*, 2016.

[19] D. Stahl and J. Bosch. Modelling continuous integration practice differences in industry software development. *Systems and Software*, 87:48–59, 2014.

[20] J. Whittle, J. Hutchinson, M. Rouncefield, H. Burden, and R. Heldal. Industrial Adoption of Model-Driven Engineering: Are the Tools Really the Problem? In A. Moreira and B. Schaetz, editors, *MODELS 2013, 16th Int. Conf. on Model Driven Engineering Languages and Systems*, Miami, USA, 2013.