Thesis for The Degree of Doctor of Philosophy

System Capability Feedback-Cycles in Automotive Software Development

S. Magnus Ågren

Division of Interaction Design and Software Engineering
Department of Computer Science & Engineering
Chalmers University of Technology and Gothenburg University
Gothenburg, Sweden, 2023

System Capability Feedback-Cycles in Automotive Software Development

S. Magnus Ågren

ISBN: 978-91-7905-791-6
Doktorsavhandlingar vid Chalmers tekniska högskola
Ny serie nr 5257
ISSN: 0346-718X

Technical Report No 231D
Department of Computer Science & Engineering
Division of Software Engineering
Chalmers University of Technology and Gothenburg University
Gothenburg, Sweden

This thesis has been prepared using LaTeX.
Printed by Chalmers Reproservice,
Gothenburg, Sweden 2023.

“Ingenjören irrar ivrigt i industrins inferno.”
- Alf Henrikson

Abstract

Context: The automotive industry is currently going through rapid change, driven by new technology; for example, electrification, autonomous driving, and connected cars. This new technology is largely based on electronics and software, and vehicles are increasingly becoming software-intensive systems. This affects how vehicles are developed, as automotive companies seek to adopt processes used in development of software-only systems, to gain the benefits of development speed and quick learning cycles possible in software development. Where sequential processes were previously the norm, automotive companies now aim to use agile methods at company-scale. Given the safety-critical nature of vehicles, and the mix software, hardware, and mechanical parts, this is challenging.

Objective: This thesis explores how system-level feedback capabilities can be achieved in development of automotive systems.

Method: To investigate a real-world setting, empirical methods are a natural choice. As an overarching research strategy, field studies are conducted at automotive companies. Over four studies, qualitative data is collected through semi-structured and structured interviews, focus groups, and workshops. The data is analyzed using adaptable methods, such as thematic coding. These qualitative approaches allow for open-ended questions, which are suitable for exploratory research.

Findings: Transitioning towards agility changes the role of architecture, requirements, and in general of system-level artifacts previously finalized during early development phases. Nevertheless, what is covered by architecture and requirements still needs to be handled. They contain accumulated expertise, and fundamental concerns, such as safety, remain. However, automotive companies need to handle an increased importance of software for new feature development. Continuing business-as-usual is not an option.

Conclusion: To achieve feedback capabilities on the system-level, there is a need for tools and methods allowing artifacts on higher levels of abstraction, for example architecture descriptions and requirements, to be modified and evolve over the entire course of development.

Keywords

Software Architecture, Requirements Engineering, Automotive Systems Engineering, Continuous Software Engineering

Acknowledgment

Firstly, my deepest thanks to my supervisors Rogardt Heldal and Eric Knauss. Without your unyielding support I could not have finished this thesis. I am also very grateful to Patrizio Pelliccione. Thank you for enthusiastic contributions to this research.

To my collaborators in the NGEA project, Anders Alminger, Magnus Antonsson, Thomas Karlkvist, and Anders Lindeborg, thank you for discussions and insights into the automotive domain.

Gösta Malmqvist, Jonas Bodén, Caroline Svensson, and Andreas Karlsson, were essential for the data collection for much of this research. I could not have done it without you.

To my office mates, Terese Besker and David Issa Mattos, you were awesome company on the roller coaster ride of PhD studies.

Lastly, to my family. Thank You!

List of Publications

Appended Publications

This thesis is based on the following publications:

[A] P. Pelliccione, E. Knauss, R. Heldal, S. M. Ågren, P. Mallozzi, A. Alminger, D. Borgentun
"Automotive Architecture Framework: The experience of Volvo Cars"
Journal of System Architecture, 77, (2017).
`https://doi.org/10.1016/j.sysarc.2017.02.005`

[B] S. M. Ågren, E. Knauss, R. Heldal, P. Pelliccione, G. Malmqvist, J. Bodén
"The Impact of Requirements on Systems Development Speed: A Multiple-Case Study in Automotive"
Requirements Engineering, 24.3, (2019).
`https://doi.org/10.1007/s00766-019-00319-8`

[C] S. M. Ågren, E. Knauss, R. Heldal, P. Pelliccione, A. Alminger, M. Antonsson, T. Karlkvist, A. Lindeborg
"Architecture Evaluation in Continuous Development"
Journal of Systems and Software, 184, (2022).
`https://doi.org/10.1016/j.jss.2021.111111`

[D] S. M. Ågren, R. Heldal, E. Knauss, P. Pelliccione
"Agile Beyond Teams and Feedback Beyond Software in Automotive Systems"
IEEE Transactions on Engineering Management, 69.6, (2022).
`https://doi.org/10.1109/TEM.2022.3146139`

Other Publications

The following publications were published during my PhD studies; however, due to contents overlapping that of appended publications or contents not being related to the thesis topic, they are not appended to this thesis.

[a] P. Pelliccione, E. Knauss, R. Heldal, S. M. Ågren, P. Mallozzi, A. Alminger, D. Borgentun "A Proposal for an Automotive Architecture Framework for Volvo Cars"
Workshop on Automotive Systems/Software Architectures (WASA) (2016). `https://doi.org/10.1109/WASA.2016.9`

[b] E. Knauss, P. Pelliccione, R. Heldal, s. M. Ågren, S. Hellman, D. Maniette "Continuous Integration Beyond the Team: A Tooling Perspective on Challenges in the Automotive Industry"
International Symposium on Empirical Software Engineering and Measurement (ESEM) (2016). `https://doi.org/10.1145/2961111.2962639`

[c] E. Johansson, T. Larsson, M. Aramrattana, P. Pelliccione, S. M. Ågren, G. Jonsson, R. Heldal "Systems of Systems Concepts for Cars"
Swedish Workshop on the Engineering of Systems of Systems (SWESoS) (2016). `http://hdl.handle.net/2077/47813`

[d] P. Pelliccione, A. Kobetski, T. Larsson, M. Aramrattana, T. Aderum, S. M. Ågren, G. Jonsson, R. Heldal, C. Bergenhem, A. Thorsén "Architecting Cars as constituents of a System of Systems"
International Colloquium on Software-intensive Systems-of-Systems at 10th European Conference on Software Architecture (SiSoS@ECSA) (2016). `https://doi.org/10.1145/3175731.3175733`

[e] Magnus Ågren, Eric Knauss, Rogardt Heldal, Patrizio Pelliccione, Gösta Malmqvist, Jonas Bodén "The Manager Perspective on Requirements Impact on Automotive Systems Development Speed"
International Requirements Engineering Conference (RE) (2018). `https://doi.org/10.1109/RE.2018.00-55`

[f] S. M. Ågren, E. Knauss, P. Giusto, G. Soremekun, R. Heldal, D. Damian "Impediments and Enablers for the Automotive Virtual Verification Ecosystem"
IEEE Software (2019). `https://doi.org/10.1109/MS.2019.2905228`

[g] P. Pelliccione, E. Knauss, S. M. Ågren, R. Heldal, A. Vinel, C. Bergenhem, O. Brunnegård "Beyond Connected Cars: A Systems of Systems Perspective"
Science of Computer Programming (2020). `https://doi.org/10.1016/j.scico.2020.102414`

[h] Regina Hebig, Truong Ho-Quang, Rodi Jolak, Jan Schröder, Humberto Linero, Magnus Ågren, Salome Maro "How do Students Experience and Judge Software Comprehension Techniques"
International Conference on Program Comprehension (2020). `https://doi.org/10.1145/3387904.3389283`

Personal Contribution

For Paper A, I participated in clarifying the scenarios, and elaborated the continuous integration viewpoint. For Paper B, I led the design of the study, carried out the interviews, led the data analysis, and led the writing. For Paper C, I participated in the architecture evaluation which created the data for the study, and led the design of the study, the data analysis, and the writing. For Paper D, I led the design of the study, carried out the interviews, led the data analysis, and led the writing.

Contents

Chapter 1

Introduction

The amount of software in automotive systems is growing at an increasing rate [Hil17]. Modern vehicles can have over 100 Electronic Control Units (ECUs), which are small computers, collectively executing gigabytes of software. ECUs are connected through several networks within the car, and the car is increasingly connected with the outside world. New capabilities such as autonomous driving and connected cars bring the automotive domain closer to the software industry. It is estimated that 80% to 90% of the innovation within the automotive industry is based on electronics [Pet14]. This more prominent role of software impacts not only the automotive systems but also how they are developed.

For pure software systems, such as web applications, current ways of working emphasize agility and continuous integration (CI) [Fow06]: Work is done in small increments, which are integrated into the whole system frequently. An automated build and test environment verifies each integration, which gives developers fast feedback on their work. This quick learning-cycle can be seen as a way to mitigate risk: by extending the software in small increments, if a problem is introduced the potential causes are typically far fewer than when large amounts of functionality are integrated at once [Fow06]. The learning-cycle can also aim for short time between the inception of a new idea and validating it with stakeholders.

Achieving comparable feedback capabilities in the development of automotive systems, and for systems with a similar mix of software, hardware, and mechanical parts, is considerably more challenging. Compared with software, development of hardware and mechanical components has long lead-times. The established practice for achieving quality is to use a stage-gate delivery process (see Section 1.1.1) dictating when integration happens. Individual components can be developed independently; however, on the whole system level, the longest lead-times determine the pace of integrations [BE15]. Thus, fast feedback cycles are confined to development isolated to single components. Shortening the feedback time also on the system-level is nevertheless an important goal within the automotive domain. The ambition is both to gain quality by resolving bugs earlier and to increase competitiveness.

Contribution This thesis pursues system-level feedback capabilities in development of automotive systems. The long-term software engineering goal is being able to continuously validate during development. Rather than first setting a fix system design, and then implementing with the assumption that a resulting implementation will be fit for purpose, the intention is to derive feedback on the system design – architecture and requirements – continuously during development. Thus, to be able to both adjust in case of unfruitful design choices, and respond to changes in external circumstances.

Outline The rest of this thesis is structured as follows. Section 1.1 describes the general background of the work; Section 1.2 covers the research method used. Section 1.3 summarizes the appended papers and Section 1.4 synthesizes the research findings. Section 1.5 discusses the validity and limitations of the research, while Section 1.6 discusses the implications of the research findings. Section 1.7 concludes the thesis introduction. After that follow as separate chapters (2 - 5) the papers constituting the thesis research. Finally, data-collection instruments used are provided in appendices A and B.

1.1 Background

Software development processes and how to organize software development has been studied since the inception of the software engineering field. To provide a background to the evolution of software development in the automotive domain, we first describe the evolution of software development processes in general, from sequential processes, via the V-model, to current agile development approaches. We then describe the specifics of automotive systems development, and the increase of software in automotive systems.

1.1.1 Sequential Development Processes

Sequential development processes model software development as a number of phases, arranged in sequence. Stage-gates between phases, where work in a later phase can only be started once a preceding phase has reached some level of completion, are common practice in sequential processes.

One early example of a sequential development process is the *waterfall process*, often attributed to Royce [Roy87], although his paper never uses the term. He observed that development projects tended to contain a set of typical phases and suggested a model sequencing these. Each phase further limits the scope of what can change. In this sense, there is a flow from abstract system requirements, through levels of increasing concretion, down to operation of the realized system, somewhat resembling the flow of a waterfall. System requirements are decomposed to software requirements, informing the analysis, in turn informing the program design, which is then implemented, tested, and finally put in operation. Figure 1.1 shows Royce's simplified model of a process divided in sequential phases.

Royce, however, cautions against sticking strictly to sequentially executing these phases, instead suggesting iterating between them. He considers it a major risk that complete system behavior is not observable until the testing

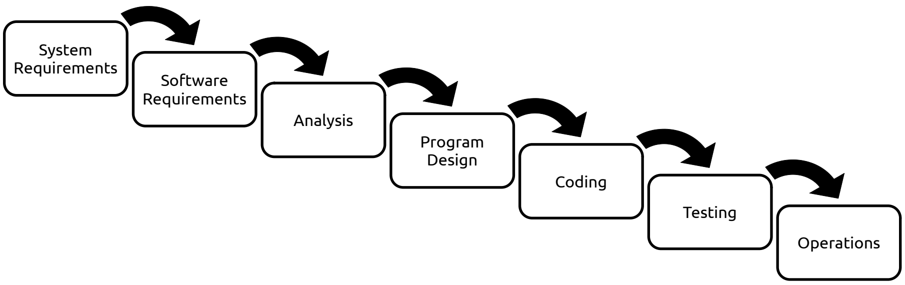

Figure 1.1: Sequential development process, adapted from Royce [Roy87]

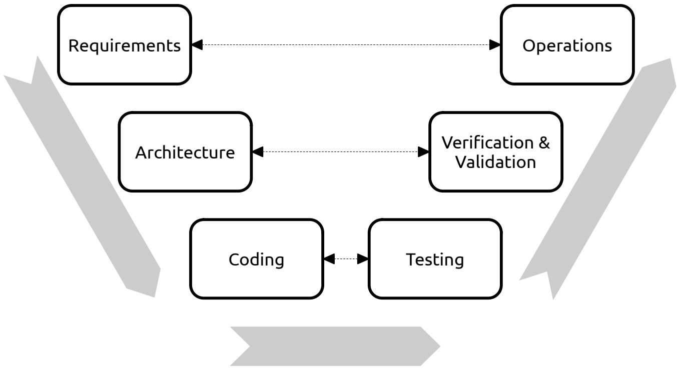

Figure 1.2: The V-model

phase. As mitigation for this risk, he proposes five strategies: 1) Start with a preliminary program design after the requirements phase, but before they have been analyzed further. 2) Document extensively during each phase. 3 Do the job twice, implement a prototype from the preliminary design. Royce primarily suggests this for cases where similar programs have not been developed before. He suggests that what eventually reaches the customer should be outcome of a second development effort, drawing on experiences from a preceeding prototype development. 4) Test extensively, with testers who have not been involved in designing the program, and visually inspect code before testing. 5) Involve the customer, before the final delivery. Standards, such as DOD-STD-2167A [oD88], that prescribed a development life-cycle similar to Royce's simplified model may thus have misunderstood his intentions.

Another common sequential development process is the *V-model*, which arranges the phases to visually resemble the letter V (Figure 1.2). The left side of the V represents decomposition from abstract requirements to concrete implementation, and the right side of the V integration to a validated system. The arrangement of phases also represents a correspondence between phases, where, for example in the validation phase, the system is validated against the system requirements. The V-model is commonly used in automotive systems development.

One issue with a sequential way of working is the ambition to do as much work upfront as possible, at the development stage when the amount of knowledge is the lowest. As part of this process, analyzing and and modeling the software to be developed was emphasized and supported by Computer-aided Software Engineering (CASE) tools. Extensive models covering the full intended system behavior was a way to create requirements with sufficient detail and coverage to drive the subsequent development phases, and avoiding uncertainty. The Rational Unified Process (RUP), marketed by IBM with an accompanying tool suite, can be seen as a culmination of this type of process. Supporting this extent of modeling and requirements engineering created a need for organizational specialization, covering these early phase artifacts. These specialist parts of the organization would deliver the requirements to the rest of the development organization. Per the process, as anticipated by Royce, the validation of these artifacts happened at a late development phase, quite some time from when they were derived, and when it was already problematic to change the built system. Sequential development process also work under the assumption that requirements will largely be stable for the duration of development. If customer expectations are volatile and change often, sequential development processes can thus struggle to handled such changes. Addressing the above problems called for new development processes.

1.1.2 Agile

Agile development was a counter reaction to processes perceived as heavyweight; requiring much effort to use and producing extensive documentation seen as *waste*. Agile development instead emphasized *individuals and interactions over processes and tools*. The notion of agile software development was coined with the publication of the agile manifesto [Hig01] in 2001, see Figure 1.3. The manifesto was the outcome of a gathering of proponents of development methods self-described as lightweight, such as Beck of eXtreme Programming (XP) [Bec00] and Schwaber and Sutherland of Scrum [Sch97]. For a full account of XP, see Beck [Bec00], for a full account of Scrum see Schwaber and Sutherland [Sch97]. Several other methods fall under the umbrella of agile, such as Kanban [And10], Lean [PP03], and Crystal [Coc04]. Lean and Kanban notably originate from automotive production [Lik04]; however, for software development in our context, the practices used are typically derived from XP or Scrum. Here we describe select practices from these methods, to illustrate how agile development intends to achieve responsiveness to change.

Continuous integration (CI) is the practice of integrating every code change directly with the latest version of the software. CI originated in XP, where, together with the practice of small releases – frequently delivering operational versions of the software under development, before all functionality is in place – they serve to get fast feedback on whether the software meets customer needs. Frequent feedback is in agile methods intended to drive responses to change: if the software is discovered to not provide utility as expected, replanning is possible throughout the development timespan. XP also originated unit tests; the idea that each piece of code, for example each method, should have an associated automated test case.

Scrum organizes development in timeboxed iterations called sprints, with typical duration of 2-4 weeks. The work to be done in a sprint is placed in a sprint backlog, a collection of work items each small enough to be completed within the sprint. During planning for a sprint, the sprint backlog is populated

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

Figure 1.3: Manifesto for Agile Software Development

from the product backlog: a prioritized list of all work items know for the system under development. The highest priority items – the top of the backlog – are the currently most important development tasks. These items are also expected to be refined to the point where development can be finished during a single sprint. Backlog refinement is done during sprints, however, Scrum takes a very general approach to what a backlog item can contain. Thus, investigations, pilot developments, and similar, can exist as backlog items, as long as their scope allows finishing within one sprint. Prioritizing the backlog is the responsibility of the product owner, which in Scrum serves the role as proxy for the customer.

Although agile development as a software development process can be interpreted in terms of concrete practices, the agile manifesto being stated as principles, represented a shift in focus from the what to the why of software development. In this view, delivering customer value after each iteration and articulating the customer value of all activities are key properties of agile. A crisp definition of agility is elusive, as Gren and Lenberg note [GL20]. Laanti et al. [LSA13] list a multitude of definitions used in research literature, and Kuhrmann et al. [KTH$^+$21] observe that agility cannot be defined solely at the process level. Gren and Lenberg propose the mitigation of operationalizing agility as responsiveness to change.

The methods originating around the same time as the agile manifesto first focused on development of systems that could be handled by single teams. The emergence of web-based software services also enabled CI and an Agile way of working. With software running on a server, not bound to a physical product provided to a customer, changes could be deployed more frequently, and with less need for planning. Where agile initially focused on a single team, as companies adopted agile for entire product development efforts, proposed methods for scaling agile arose. Examples include the Scaled Agile Framework (SAFe) and Large-Scale Scrum (LeSS). For this thesis, we only consider SAFe, as this framework in particular was considered by our case companies (see 1.2).

The Scaled Agile Framework was developed by the originators of Scrum. SAFe takes concepts from Scum, notably sprints and backlogs, and adds mechanisms for coordinating the pace between multiple teams. A number of teams working in aligned cadence is termed an Agile Release Train (ART).

The sprint duration and starting points are aligned for the ART. A set number of sprints, typically five, are termed a Program Increment (PI). Between each increment the ART simultaneously and collectively plans the next PI. Work items spanning the ART are prioritized in a program backlog. During the PI planning the program backlog items are broken down into smaller items, which are then placed in prioritized team backlogs, representing the work a team will deliver during the next PI. While work items covering architecture and requirements are part of the program and team backlogs, the roles of program management and systems architecture reside outside of an ART. This particular aspect of integrating architecture and requirements in agility at scale is revisited in the thesis results. The typical size intended for an ART is 50 to 125 people. For organizations larger than that, SAFe uses a similar concept of aligned cadence to coordinate multiple ARTs

1.1.3 Automotive Systems Development

Historically, automotive manufacturers have used sequential development processes, such as the V-model. Reliance on rigid requirements has been used both as a way to ensure that the vehicles developed are safe, and as a way to divide development among multiple tiers of suppliers. Original Equipment Manufacturers (OEMs) – the companies behind automotive brands, and typically the owners of vehicle factories – have initiated development by contracting suppliers to develop components in accordance with requirements specifications. The OEMs have then focused on integrating deliverables, often in the form of electronic or mechanical components.

With the increase of software in automotive systems, the integration of software has increased, where integration of components was previously mainly integration of mechanical or electrical interfaces. That innovative functionality, such as autonomous driving and connectivity features, are driven by software also push the automotive industry closer to consumer electronics, particularly in terms of customer expectations. For OEMs, this drives a need to increase responsiveness to market changes, and shorten the time for development of new models. If a sequential development process is used, with requirements being created in an early phase, and the full development time is several years, as may be the case for a new vehicle model, a software-only function, such as a voice assistant, developed against those requirements, will be obsolete already once it reaches the market. This particular example was brought up during the study in Paper D, see Section 5.4.1 for further detail.

In this circumstance, the responsiveness to change promised by agile methods is alluring. Automotive companies are also pursuing agile methods, however, automotive systems have a number of properties that hinder straightforward application of agile methods. The system scale itself means that some method of scaling agile is needed. Coordinating the development also calls for systems architecture work, which was originally down-prioritized in agile methods. Vehicles fundamentally being physical products also means that development must cover a mix of hardware and software. The fundamental vehicle functionality, such as controlling the brakes and engine, is software intensive, but the software implements control-loops close to the hardware; considerably different from software for web services. Furthermore, autonomous driving, for example, utilizes other vehicle functions, as braking and environment sensing. Functions combining other functions on the vehicle level adds not only more software but also complexity to the entire vehicle. A large part of the software in a vehicle is also safety-critical, which comes with legal requirements and adherence to safety, which call for development processes to be precisely describe and possible to inspect. Neither process descriptions, nor the ability to inspect process adherence, where originally any focus of agile methods.

The research in this thesis studies the automotive domain as this shift towards agile ways-of-working is underway. In particular, we focus on the system-level challenges.

1.2 Method

As the context of this research is the real-world automotive domain, empirical methods are a natural choice. Having access to a real-world context, and to collect data from that context, is what primarily informs the choice of methods for this thesis.

1.2.1 Research Context

The shared context of the research is the automotive domain; specifically, the research has been carried out in the Vinnova FFI[1] research project Next Generation Electrical Architecture (NGEA). The overall purpose of the NGEA project was to get scientific input for use in an ongoing effort at the project lead partner to develop their next vehicle architecture generation. To this end, the project investigated a broad number of topics, including: architecture evaluation, architecture topologies, cars as constituents in a system-of-systems, the automotive standard framework AUTOSAR Adaptive [FB16], transparency between collaborating organizations, patterns and strategies for continuous integration, and development speed and feedback. We participated across the project activities. The research in this thesis falls specifically under the topics of architecture evaluation, and development speed and feedback. Each paper covers in further detail the specific context for that particular study. Here, we cover the context shared between them.

Through the NGEA project, we had access to topic experts from automotive OEMs, suppliers, and research institutes. The project consortium consisted of a number of partners, with different roles in the automotive domain, Table 1.1 gives an overview. The project was led by key architects from OEM. Other partners sent key people (architecture and representatives of business parts of their organizations) to react to this. Via one of the project partners, we also had access to one additional OEM.

The project was organized in the form of workshops and meetings held on-site at the project partners. Sessions were frequent, on average one to two sessions per week. Beyond the scheduled sessions, we also had access to visit company premises of the project lead partner, allowing for informal discussions. The project was thus situated in the everyday development work, rather than

[1]FFI (Fordonsstrategisk Forskning och Innovation – Vehicle-strategic Research and Innovation) was a research program at Vinnova, the Swedish innovation agency.

Table 1.1: Project Partners

Partner Id	Role and expertise provided to the project
P1	One OEM, leading the project and providing key architects and a vehicle-level perspective.
P2 – P4	Three suppliers of electronics and software, providing knowledge on specific technologies, for example Adaptive AUTOSAR and safety-certified components.
P5 – P10	Six technology consultant suppliers, providing knowledge on specific topic areas, for example continuous integration, testing, and agile methods.
P11 – P12	Two universities, providing a research perspective on software development in general, and specifically on architecture, requirements, and system-of-systems.
P13 – P16	Four research institutes, providing research perspectives on specific topics, for example safety and system-of-systems.

performed in an isolated context. At the start of the project, there were no established practices for feedback-loops for the full scale of automotive product development; evolving such methods was a key concern of the project. We entered at a point in time when the project partners were pursuing this, but did not have it fully in place.

1.2.2 Research Strategy

Stol and Fitzgerald, citing McGrath [SF18, McG81], observe that all study design involve tradeoffs between generalizing over actors (A), precisely measuring their behavior (B), while maintaining a realistic context (C). Different research methods optimize differently between the factors A, B, and C. Which optimization, and consequently which method(s) to choose, depends on the study goal. In their ABC framework, named for the aforementioned factors, Stol and Fitzgerald categorize research methods in different research strategies, positioned against the dimensions of obtrusiveness and generalizability. Obtrusiveness refers to the extent to which the research (or the researcher) manipulates or instruments the research setting. Generalizability refers to how specific to the research context the results are.

Field studies are a research strategy offering maximum potential for context realism. Studies, and their findings, are context specific, while the room for affecting the context is limited. The realism gained thus come at the expense of low precision in measurements of behaviors, and low generalizability of the findings. In contrast, the laboratory experiment strategy offers precise measurements at the expense of realism. A common method within the field study strategy is the case study [RH09]. To fully utilize the real-world context, this thesis uses the field study as the strategy for all included studies.

Stol and Fitzgerald distinguish between knowledge-seeking and solution-seeking studies, where knowledge-seeking studies aim to learn something about the world, whereas solution-seeking studies aim to solve practical problems. The ABC framework focuses exclusively on knowledge-seeking studies, which is an aspect of all research included in this thesis. Two of the included studies also include solution-seeking aspects. These aspects of the research is design-oriented, as termed by Piirainen and Gonzalez [PG13]. Design-oriented research seeks to solve practical problems by constructing artifacts. The scientific contribution stems from the problem having academic relevance, the artifact providing a novel or innovative solution to the problem, and the solution having practical relevance. To this end, utility-based evaluation – that the constructed artifact is deemed useful by practitioners – is a common form of validation.

1.2.3 Research Objective

The research in this thesis contributes to the following objective:

> *Achieving system-level feedback capabilities in development of automotive systems.*

The research was conducted as four separate studies, covering different aspects of system-level feedback, and thus together contributing to the thesis objective. The studies were published as independent papers. Section 1.3 describes the specific focus of each study.

Paper A Automotive Architecture Framework: The experience of Volvo Cars

Paper B The Impact of Requirements on Systems Development Speed: A Multiple-Case Study in Automotive

Paper C Architecture Evaluation in Continuous Development

Paper D Agile Beyond Teams and Feedback Beyond Software in Automotive Systems

1.2.4 Data Collection and Analysis

In line with the field study strategy, data was collected in the field, at partners in the NGEA project. Respondents participated in our studies either as focus group participants or as interviewees. We selected respondents from technical specialist roles, for example architects, and manager role. This complements the developer role perspective in previous studies [KLK+17, ADW17, ISM+15, HDLP15, EHKP15] conducted in comparable contexts. The respondent selection was thus purposive, that is, respondents were selected for their specific perspective.

For Paper A, data collection and analysis were interleaved. Through a number of focus groups, we drafted and refined the artifacts proposed in the paper (architecture framework viewpoints). This thesis uses the term focus group, rather than group interview or workshop, to denote a session where a group (of one or more researchers and one or more respondents) collaboratively discuss and refine a work item; an artifact of some sort. While the research in Paper A was conducted in a field setting, it differs from the other papers included in this thesis by being more solution-seeking than knowledge-seeking, in the terms used by Stol and Fitzgerald [SF18].

For Paper B, data collection was done in two stages. First, a series of semi-structured interviews was conducted, with the interviews recorded, transcribed, and analyzed. We used thematic coding [Sal15] to analyze the transcripts; to focus the analysis, we used a lens of requirements engineering. The findings from the first round of interviews were then validated through a second interview series. Since we focused this round of interviews more on validation than exploration, we used a structured form, primarily with Likert scale answers.

In Paper C, we did a post-hoc analysis of data collected throughout the NGEA project. During the second half of the project, we pursued architecture evaluation, with the Architecture Tradeoff Analysis Method (ATAM) as starting point for the evaluation process. We took part in the evaluation as participant observers. Work items, such as whiteboard sketches, notes, and slide decks, created while conducting the evaluation, evaluation reports, and experience notes taken, were analyzed after the evaluation efforts had concluded. The data thus shares a trait with archive data, that it was created for a different purpose than the study in which it was analyzed. We refined our draft findings through a series of focus groups.

Paper D revisited the data collected in the first stage of Paper B. Where Paper B used only those parts of the data that pertained to requirements; approximately 15% of the data, in Paper D the whole data set was analyzed. In contrast to the analysis in Paper B – which first extracted the parts of data, and then coded only those – here we took a specific lens of three research questions and analyzed the full data set, again using thematic coding [Sal15], selectively coding fragments relating to our lens. The pervading concern in our lens is an agile vehicle-level feedback loop beyond individual teams.

For the thesis in all, the data are thus qualitative, in the form of interview transcripts, categorical data (Likert scale), notes and working materials from workshops. For our purpose of exploration, such flexible form data is a good fit, as it allows for open-ended questions. Given this open form of data, we have also relied on adaptable analysis methods, as thematic coding.

Papers A and C are similar in regards to how an artifact – viewpoints for an architecture framework or principles of continuous architecture evaluation – was refined in focus groups. Data was collected using workshops, focus groups, and interviews, both semi-structured and structured.

To further ensure grounding in the context, all studies are done in collaboration with industry.

1.3 Paper Summaries

The study design and contribution of each included paper is summarized below.

1.3.1 Paper A: Automotive Architecture Framework: The experience of Volvo Cars

The architecture is a key instrument for handling the complexity of automotive systems development, and to balance the multiple, possibly conflicting, concerns involved. When working on some specific aspect, it may be helpful to view the system under development from a specific viewpoint; fitting every concern in one

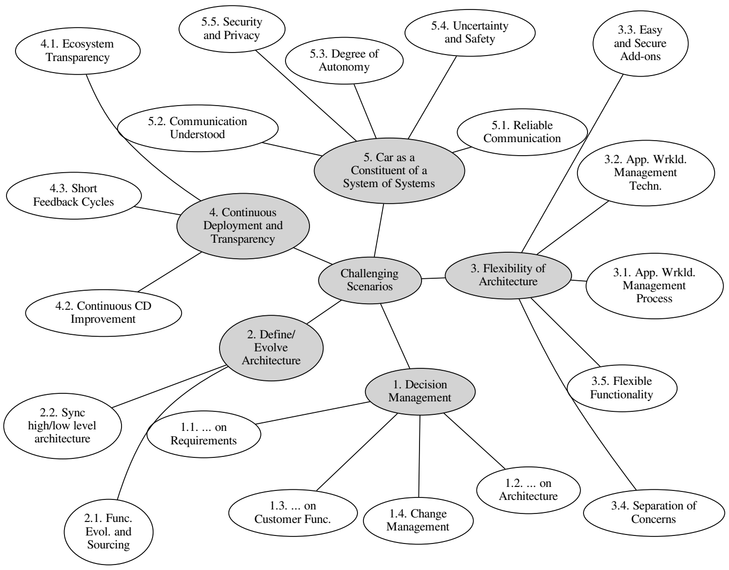
Challenging Scenarios
1. Decision Management
1.1. ... on Requirements
1.2. ... on Architecture
1.3. ... on Customer Func.
1.4. Change Management
2. Define/ Evolve Architecture
2.1. Func. Evol. and Sourcing
2.2. Sync high/low level architecture
3. Flexibility of Architecture
3.1. App. Wrkld. Management Process
3.2. App. Wrkld. Management Techn.
3.3. Easy and Secure Add-ons
3.4. Separation of Concerns
3.5. Flexible Functionality
4. Continuous Deployment and Transparency
4.1. Ecosystem Transparency
4.2. Continuous CD Improvement
4.3. Short Feedback Cycles
5. Car as a Constituent of a System of Systems
5.1. Reliable Communication
5.2. Communication Understood
5.3. Degree of Autonomy
5.4. Uncertainty and Safety
5.5. Security and Privacy

Figure 2.2: Overview of challenging scenarios (replicated here for convenience)

single representation may also be infeasible. Dividing architectural description of a system into separate parts for different concerns leads to the need for coordinating these, however. A state-of-the-art approach to coordinating multiple architectural views is an architecture framework. The ISO/IEC/IEEE 42010:2011 standard defines an architecture framework as *"a coordinated set of viewpoints, conventions, principles, and practices for architecture description within a specific domain of application or community of stakeholders"* [ISO11]. A viewpoint, and the associated notion view, is defined by the standard as *"A viewpoint is a way of looking at systems; a view is the result of applying a viewpoint to a particular system-of-interest"*. An architecture framework viewpoint thus addresses particular stakeholder concerns.

In Paper A, we propose extensions to the state of the art on automotive architecture frameworks in the form of three viewpoints: (i) *Continuous Integration and Deployment*, (ii) *Ecosystem and Transparency*, and (iii) *System of Systems: vehicle point of view*. The proposed viewpoints derive from stakeholder concerns, captured in the form of challenging scenarios, elicited through a a series of focus group sessions (see Figure 2.2).

Both the selection of these particular viewpoint, and the concerns they address, stem from the full set of scenarios. For this thesis, however, relating to the research objective, the focus is on the *Continuous Integration and Deployment* viewpoint. Scenarios that particularly motivate this viewpoint include *The need for establishing short feedback cycles*; developers need quick feedback on how their contributions will work on each level of integration. *Easy and secure add-ons*; to avoid a big-bang integration just before the start of production, developers need the ability to add functions even after the vehicle has been put into use.

The CI viewpoint aims to address two questions: 1) How do continuous integration and deployment practices in automotive engineering impact the system architecture? and 2) How do architectural decisions in the system architecture impact continuous integration and deployment practices? The concerns covered by the CI viewpoint – for example, quick response to market changes, handling resulting architecture changes, and ensuring that an architecture is fit for purpose – point towards a need for feedback capabilities on the architecture. The concerns also point towards a need for continuous architecting, which is explored further in Paper C.

1.3.2 Paper B: The Impact of Requirements on Systems Development Speed: A Multiple-Case Study in Automotive

Given the stable, established, engineering practices of the automotive industry, moving toward CI [KPH^{+}16] and large-scale agile methods [Lef16] leads to challenges in how to work with requirements [SKV10, KKNK17]. As described above, increasing development speed is a key reason for why automotive companies are seeking to change ways-of-working. At two automotive Original Equipment Manufacturers (OEMs), both having agile transformation initiatives ongoing to pursue such change, we investigated the following research questions:

RQ1: Which aspects of the current way of working with requirements impact development speed?

RQ2: Which new aspects should be considered when defining a new way of working with requirements to increase development speed?

RQ3: To what extent will either aspects be addressed through the ongoing agile transformation?

We conducted a multiple-case study design organized in two steps, with the two OEMs as cases. In the first step, our respondent selection focused on manager roles, complemented with technical experts in processes and architecture. As a complement to previous works covering a developer perspective [HDLP15, ISM^{+}15], we thus acquire the perspective of managers, who in their roles of dividing and leading work rely on requirements. For data collection, we used semi-structured interviews, with 20 respondents; each interview lasting approximately one hour. From a thematic coding of the data, we derive six themes in answer to RQ1, and six in answer to RQ2, see Figure 3.1 In the second step, we validated our results from the first step, and relate those findings to RQ3. We conducted 12 additional interviews, this time structured interviews, using mainly Likert scale questions. Nine of the second step respondents were new to the study.

Regarding RQ1, we discovered that what has historically been the way-of-working still plays a crucial role. Emphasis is still on decomposition of requirements and many levels of abstraction. Rather than pursuing development speed, in the sense of enabling changes when feedback on decisions becomes available, processes are forcing early design decision.

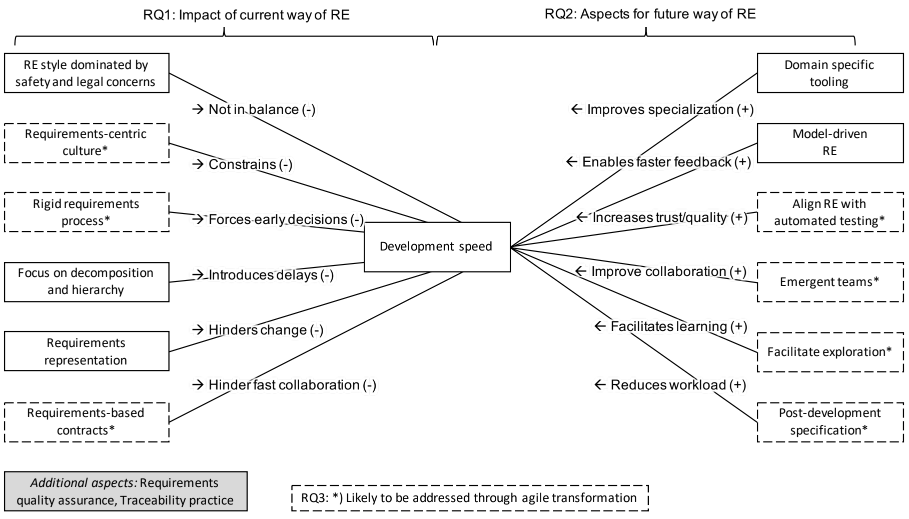
RQ1: Impact of current way of RE
RQ2: Aspects for future way of RE
RE style dominated by safety and legal concerns
Requirements-centric culture*
Rigid requirements process*
Focus on decomposition and hierarchy
Requirements representation
Requirements-based contracts*
→ Not in balance (-)
→ Constrains (-)
→ Forces early decisions (-)
→ Introduces delays (-)
→ Hinders change (-)
→ Hinder fast collaboration (-)
Development speed
← Improves specialization (+)
← Enables faster feedback (+)
← Increases trust/quality (+)
← Improve collaboration (+)
← Facilitates learning (+)
← Reduces workload (+)
Domain specific tooling
Model-driven RE
Align RE with automated testing*
Emergent teams*
Facilitate exploration*
Post-development specification*
Additional aspects: Requirements quality assurance, Traceability practice
RQ3: *) Likely to be addressed through agile transformation

Figure 3.1: Themes (replicated here for convenience)

Regarding RQ2, a number of technology-oriented improvements are proposed: domain-specific requirements tooling, model-based rather than text-based requirements, and test automation. In addition to these, organizational improvements are also proposed. Separation, intended to achieve team autonomy and clear divisions of responsibilities, can instead lead to slow workflow, extensively based on handovers. Instead of the extensive up-front specification needed to support such organizational separation, respondents suggest combining a lightweight pre-development requirements engineering approach with precise specifications created post-development.

Regarding RQ3, our respondents provide diverse view (see also Figure 3.4). Although, transforming the way-of-working to agility is regarded as addressing many of the identified impacts on development speed, the responses indicate that the transformations do not fully cover neither impacts of current ways-of-working nor new aspects. One reason is that these issues are already getting attention independently of the agile transformation. Pursuing agile ways-of-working can thus not claim all the credit. Most aspects of the desired future way-of-working with requirements have been reported to work well in isolation. However, their interplay is not sufficiently clear (as for example safety concerns and continuous integration or deployment at system level).

1.3.3 Paper C: Architecture Evaluation in Continuous Development

In sequential development processes architecture is created mainly during an early phase and then used to guide subsequent development phases. Agile development methods, however, where the implementation evolves continuously, changes the role of architecture. In Paper C we investigated how architecture evaluation can provide useful feedback during development of continuously evolving systems, operationalized as the following research questions:

RQ1: How can architectural evaluation in a continuous setting provide timely feedback on whether a specific capability is supported by the current architecture?

RQ2: What information should be provided to support architectural evaluation in a continuous setting?

RQ3: How in principle can existing frameworks for architectural evaluations be made fit for continuous evaluation?

As the starting point for how to organize the evaluation, we used the Architecture Tradeoff Analysis Method (ATAM); a scenario-based architecture evaluation method. Scenario-based methods express quality attributes (for example flexibility and maintainability) as scenarios, describing the response of the studied system to a certain stimulus. Given the uncertainty inherent in investigating future architecture needs, a scenario-based method was promising, as these *"estimating risks and uncertainty associated with the systems' requirements"* [IHO02, p. 11]. A brief description of scenario-based methods and a summary of ATAM is given in Section 4.2.

The architecture evaluation efforts took place both in the NGEA project and at the OEM leading the project (partner P1, see Table 1.1). In the NGEA project, one of the workgroups was tasked with bringing together the output from the rest of the project. The goal was to evaluate proposals from the other workgroups, analyze tradeoffs between them, and from this derive recommendations for future electrical architecture. This prompted the initial question of how to provide feedback on what capabilities a particular architecture supports. The architecture evaluation efforts spawned work throughout the NGEA project and at P1, resulting in documents and other work items, such as whiteboard sketches. Through a post-hoc analysis of these, we derived an initial set of principles for continuous architecture evaluation. We then refined and validated these through a series of focus group sessions, arriving at four principles, see Table 4.4.

Table 4.4: Principles of Continuous Architecture Evaluation (replicated here for convenience)

CAP I	Evaluate decisions on demand, in response to a clear stakeholder question
CAP II	Evaluate architectural decisions incrementally to manage evaluation scope
CAP III	Evaluate in the context of the full integrated product to support incremental architecture evaluation
CAP IV	Apply concepts of evaluation constructively to articulate the rationale for an architectural decision

The research questions are answered by the interplay of the principles. For RQ1, the principles combined support incremental evaluations of narrow scope that fits into the tight schedule of continuous development practices.

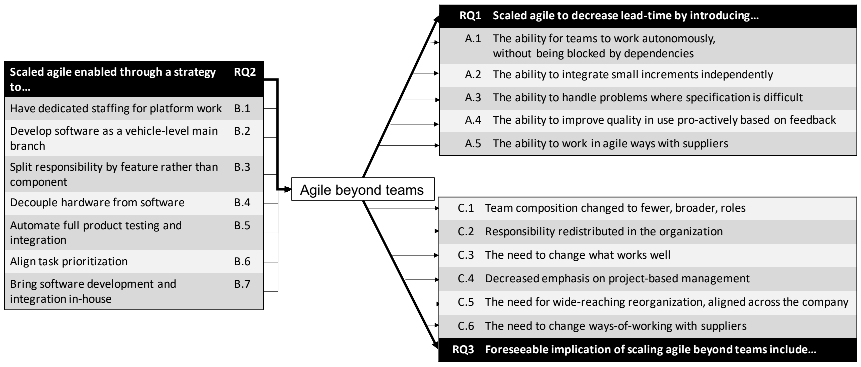
RQ1 Scaled agile to decrease lead-time by introducing…
A.1 The ability for teams to work autonomously, without being blocked by dependencies
A.2 The ability to integrate small increments independently
A.3 The ability to handle problems where specification is difficult
A.4 The ability to improve quality in use pro-actively based on feedback
A.5 The ability to work in agile ways with suppliers
Scaled agile enabled through a strategy to… RQ2
Have dedicated staffing for platform work B.1
Develop software as a vehicle-level main branch B.2
Split responsibility by feature rather than component B.3
Decouple hardware from software B.4
Automate full product testing and integration B.5
Align task prioritization B.6
Bring software development and integration in-house B.7
Agile beyond teams
C.1 Team composition changed to fewer, broader, roles
C.2 Responsibility redistributed in the organization
C.3 The need to change what works well
C.4 Decreased emphasis on project-based management
C.5 The need for wide-reaching reorganization, aligned across the company
C.6 The need to change ways-of-working with suppliers
RQ3 Foreseeable implication of scaling agile beyond teams include…

Figure 5.1: Themes with respect to the three research questions (replicated here for convenience)

With respect to RQ2, we found quality attribute scenarios to provide a way to constructively articulate the value an architectural decision provides, and to connect architectural decisions to business goals. For RQ3, based on our principles, we give recommendations for what to consider when conducting architecture evaluation in continuous settings.

1.3.4 Paper D: Agile Beyond Teams and Feedback Beyond Software in Automotive Systems

Although Agile ways-of-working initially focused on small software development teams, the success of agile approaches has led to adoption in different contexts, such as automotive. In Paper D, we investigate scaling agility beyond single teams. In particular, we focus on the aspect of fast and early feedback on the product level during development. We investigate what companies are seeking to achieve, how they envision doing it, and what implications they foresee from such drastic organizational change. As a lens for analysis, we operationalize this as three research questions.

RQ1 What abilities are automotive OEMs seeking to achieve through an agile vehicle-level feedback loop beyond individual teams?

RQ2 How are automotive OEMs proposing that an agile vehicle-level feedback loop beyond individual teams can be established?

RQ3 What implications do automotive OEMs foresee from the proposed ways of introducing an agile vehicle-level feedback loop beyond individual teams?

This paper is based on the same data set used in Paper B, but where Paper B uses only those parts of the data that pertain to requirements; approximately 15% of the data, in Paper D the whole data set is analyzed. Our sampling strategy is described above for the first step of the study in Paper B. We use thematic coding to analyze the data, and derive a number of themes for each research question, see Figure 5.1.

For RQ1, our respondents suggest that working in an agile way also at the organizational scope that involves the entire vehicle could mean establishing a short feedback loop, where teams deliver small increments, increments are integrated with the vehicle, and the team receives feedback fast enough to match the typical pace of agile sprints of a few weeks. The intention being that increments can be both delivered and integrated independently, keeping a consistent flow out from each team. The ability to validate or refute assumptions close in time to when they were made is also seen as way to reduce dependency on specifications, and allow for exploration during development.

For RQ2, a common thread in respondents' suggestions is to establish a platform or notion of vehicle-level software. Teams would integrate small increments into this throughout development, rather than providing large deliverables for full vehicle integration only at a few occasions.

Lastly, for RQ3, respondents note that introducing the sought feedback loop may necessitate changes to currently well-performing ways-of-working. Although a feedback-driven way-of-working may increase throughput and team autonomy, it can also also increase ambiguity of where in the organization responsibility resides, and decrease the emphasis on precise specification and documentation. Additionally, the complexity of interdependent parts in the vehicle is not automatically handled any different because a vehicle-level feedback loop is introduced. A further example is that going from a focus on projects to a focus on sustaining development of a shared platform impacts when returns on investments can be observed and where in the organization prioritization between different possible features happens.

1.4 Synthesis

Throughout the research, the topics in focus shifted, from architecture to the implications of architecting. Architecture and requirements are not topics prominently treated in agile canon, while in the automotive domain they have key roles. When the domain transitions towards agility, what is covered by these key concepts still needs to be handled; throwing away expertise or neglecting fundamental concerns such as safety is not an option. However, the shifts instigating agile transformations also preclude business as usual. Automotive companies need to handle an increased importance of software for new feature development.

The work in Paper A, extending an architecture framework with viewpoints, served as a starting point. The concerns addressed by the viewpoints spurred further investigations. Van der Valk et al. continued research in the direction of the ecosystem and transparency viewpoint [vdVPL+18]. Works continuing in line of the system-of-systems viewpoint include Johansson et al. [JLA+16], Pelliccione et al. [PKL+16], and Pelliccione et al. [PKÅ+20]. This thesis continues from the continuous integration viewpoint. The main contribution of Paper A to this thesis is thus the concerns framed by the CI viewpoint, rather than the viewpoint as an instrument for software development. Among these concerns were quickly responding to change. Prompted by this, we investigated the impact of responsiveness, in terms of development speed, on architecture and requirements.

Development speed as considered in Paper B mainly deals with how requirements work impacts the system development speed, but also contains the speed of feedback on requirements. For reasoning about system capabilities, requirements are key for safety and legal aspects. Requirements as basis for contracts, however, can be a hindrance to collaborative development setups. We find that the factors inhibiting speed, and hence hampering responsiveness, are among what our respondents consider addressed, although not unanimously, by a move to agile methods (see Figure 3.4), but more than ways-of-working are involved. Technical solutions, such as supplementing textual requirements with model-based ones, need to be combined with organizational solutions.

Paper C then investigated responsiveness on architecture. Specifically, how to make feedback on an architecture manageable in an agile setting. During continuous development, the readily available feedback needs to cover the capabilities of an architecture in use. This makes evaluation methods with extensive scope unwieldy. We propose principles to guide architecture evaluation in continuous settings. In addition to limiting the scope, demand from the organization, and constructive use of evaluation concepts, are organizational aspects we identified. On the technical side, we see the need for a representation of the entire product, which partial evaluation results can be integrated to.

In Paper D, we then look broadly on responsiveness in terms of feedback-cycles beyond individual teams. Taking the findings from the four papers together, suggest that achieving system-level feedback capabilities, as a basis for responsiveness, involves dependencies between organizational and technical solutions. One approach to making system-level feedback continuously available, that is, possible during the daily development at any time, would be to pursue system-level understanding in terms of the lower level implementation. In other words, how the implementation affects and realizes the properties and capabilities of the system. To this end, evaluating representations of the system-level – the architecture description – is one mechanism. A platform approach is a technical strategy in the implementation for enabling this. Platform work, however, may require dedicated staffing; and how work and responsibilities are distributed across the organization may also need to change.

1.5 Analysis of Validity and Limitations

This section discusses the validity and limitations of the thesis research as a whole. To structure the validity analysis, we follow the scheme proposed by Runeson and Höst [RH09].

1.5.1 Construct Validity

Construct validity reflects *to what extent the operational measures that are studied really represent what the researcher have in mind* [RH09]. To achieve this, we have immersed ourselves in the domain, through the NGEA project, to enable a shared, common, understanding of concepts, terms, problems, and constructs, with respondents. We also had industry collaborators participating as co-authors throughout study design and execution.

We leveraged this knowledge when constructing study instruments. Discussions in the project informed the choice of questions and topics. Refinement was then done through multiple iterations, with input from senior industry experts. When collecting data, through NGEA we had access to collaborators working at the case companies We could thus contact respondents in a context already part of their work, which allowed for informal interactions, for example during interviews, and openly sharing information. With industry co-authors, we also got input during the data analysis. Furthermore, preliminary results were presented and discussed in the project consortium.

1.5.2 Internal Validity

Internal validity *is of concern when causal relations are examined. When the researcher is investigating whether one factor affects an investigated factor there is a risk that the investigated factor is also affected by a third factor* [RH09]. In other words, internal validity concerns whether study results can be explained by the claimed factors, or if other mechanisms are at play.

With agile transformations underway, respondents may react to the change process, rather than then contents of the pursued ways-of-working. This could give a positive inclination towards agile as forward-looking and implicitly better, or a negative reaction to change. Despite making an effort to ask questions neutrally, respondents might be biased with respect to their current ways-of-working. More broadly, when researching a forward-looking topic, such as proposed improvements to ways-of-working, ideas of potential improvements may receive prominence over practices that currently work well.

Much agile literature originates from proponents of agile ways-of-working. That material on agile largely paints a positive picture of agility could influence perceptions of both researchers and respondents. As mitigation, we spent time refining interview and focus group instruments, considering the phrasing of the questions and types of questions to avoid altogether. For example, when asking about potential improvements to development speed, we controlled for whether respondents considered the proposed improvement worthwhile to spend time on. Notably, respondents were not universally, uncritically positive towards agile. Sections 3.8, 4.7, and 5.5 discuss interpretations of the results and their implications specific to each study.

1.5.3 External Validity

External validity concerns *to what extent it is possible to generalize the findings* [RH09]. For qualitative studies, generalizability is inherently confined to being analytical, as there is no statistical generalization to a population. All research in this thesis was conducted within the context of the NGEA project. Participants thus had a shared frame of reference, which influences the results. The results are also specific to the automotive domain, in one country, and to one software lifespan. Furthermore, by the overall field study research strategy of this thesis, realism of context comes at the expense of the possibility for generalization [SF18].

The closest analytical generalizations would be automotive companies in general. For example, if studying adoption of agile ways-of-working at other automotive companies, the factors we identify are plausible hypotheses for what might be in play also in a largely similar context. Additionally, the pursuit of development speed and fast feedback is not exclusive to the automotive domain. The obstacles and circumstances are particular to the studied case, however.

1.5.4 Reliability

Reliability concerns *to what extent the data and the analysis are dependent on the specific researchers* [RH09]. As discussed for construct validity, we were fortunate to have access to practitioners in the domain, having frequent interactions over several years. This a resource with can be difficult for researchers to acquire. The OEM driving the project also has long-standing collaboration with academia. Hence, they are familiar with the difference between industry and academia, and have experience in choosing what can be shared. Fundamentally, this happens at project creation, when assessing if the topic is one where collaboration is deemed possible. Hence the research has access to the current front of the domain. However, it's not clear whether selective omissions have influenced the results; each respondent judges what can, or cannot, be shared. While the topics this thesis researches are directly involved in the development work at our case companies, the results are not in themselves product content, and thus not an immediate aspect where the companies compete. Altogether, the results are thus more particular to the specific case than the specific researchers.

Throughout the studies, we involved several researchers for each aspect. During interviews, we used multiple interviewers. During focus group sessions and meetings, we also had more than one researcher participating in each session. This allowed for subsequent discussions and combining different perspectives. Data analysis and subsequent writing was also done as a collaborative effort.

1.5.5 Limitations

As noted in the validity analysis, the results are specific to the automotive domain, which is a limitation of this thesis. Studies also focus on software-intensive parts of automotive systems development. As vehicles contain extensive amounts of software, although the domain is shared, there is room for variation within. For example, development of graphical interfaces for an infotainment system, processing of map data for navigation, and implementing engine control loops each have their respective uniqueness. The domain reliance on sequential development process, and emphasis on requirements nevertheless permeates the development. As does the ongoing transformation to achieve more effective feedback loops.

Studies were conducted at a given point in time, and at particular point of transformation in the automotive domain. Whether automotive-specific factors dominate, or if increased software-intensity in another domain would be dominated by similar factors is an open question.

Respondents of specific roles were selected for their particular perspective. This approach has inherent limitations of generalizability similar to those of case studies, that something specific is investigated, here that specific persons participate as representatives of their roles. Their perspective may hold for their role more broadly, but this analytical generalization is speculative, and would need to be validated in each specific other case. Our focus on management roles also means that developer-focused aspects of agility are less prominent in our findings. The goals for pursing agility center around time-to-market and being able to quickly adapt a product, rather than team well-being. This might be a side-effect of which questions we asked and how we framed the research, but it is not an aspect we explored further.

1.6 Discussion

Although specific to automotive domain, our research raises discussion points with possible implications also for software development in general.

1.6.1 Critique of Software Engineering

Transitioning to agile ways-of-working in automotive development has not lead to large and frequent changes in development scope becoming the norm. Vehicle development is still product development where lead-times of hardware and mechanical components call for pre-planning. Working in an agile way might, however, over time change the expectation of how thoroughly software development can be planned in advance. Responsiveness to change is one ability our case companies are seeking to gain from agility. Achieving this may require expecting less certainty beforehand from software development. More certainty requires more planning, which limits the flexibility. Gren and Lenberg [GL20] frame agility as responsiveness to change. Contrasting agile with lean, they describe the latter as efficiency (doing things right), and the former as effectiveness (doing the right thing). In other terminology, this could be seen as verification and validation. Agile methods, in this sense, focus on the validation of the software developed; trading precise planning of the scope of work for frequent feedback on whether the work done provides utility to the stakeholders. Plan-driven methods focus on verification, that the software developed meets its specification, irrespective of how well the specification captures stakeholder utility.

Perhaps there's reason to be critical about the perceived nature of software development, for example how possible it is to go from prescriptive high-level abstractions to implementation. Since its inception, the notion of software engineering has been subject to criticism. Could it be that the software crisis – the problem of writing useful and efficient programs in a required time – is rather a management crisis? This critique seems to recur with some regularity in software engineering. Ensmenger [Ens12], in "The computer boys take over", gives a historical view of how the software profession was codified during the 1950s and 60s, and how management felt ousted by these *computer boys*. Management would here be interpreted as a quest for controllability of software development, rather than validity (i.e. utility) of the resulting software.

Mahoney [Mah04] also gives a historical account of the field of software engineering and how the reasoning has stuck close to traditional manufacturing. Dijkstra [Dij88a, Dij88b] wrote about the *radical novelty* of software; arguing that the digital (discrete) nature of software, unlike the analog (continuous) nature of mechanical components, is unsuitable to tackle with traditional engineering. Garman [Gar81] in 1981, writing about a bug in the space shuttle, notes that

> *[T]he project is assumed to know what it wants out of the software in the beginning! Software development from my perspective is almost never that way. It should be, and we should try hard to make it so – but it is always iterative and incremental instead.*

He called for more research on how to reliably modify software, with minimum impact on time and cost. Our findings provide guidance on evolving an architecture continuously during development. However, how to influence the architecture of a large-scale system from the level of its implementation needs further research.

1.6.2 Meta-level Process and Empiricism

Sequential development processes might be precisely prescribed ways-of-working. Agility might instead be a meta-level approach, where how to work is regularly reconsidered. One interpretation of the twelfth principle of the agile manifesto [Hig01] – *At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly* – is that, in order to respond to change, the team regularly reflects on, modifies, and adapts its way of working. This could imply a need to reconsider what architecture and development of continuously evolving systems is. Ralph [Ral18] contrasts empiricism against rationality. Arguing in favor of the former, he proposes that there is no *"the problem"*. Instead of a clear problem for which developers create a solution, he argues that developers and stakeholders collaboratively construct a desired system.

> *Each project presents unique sequences of events, which do not necessarily resemble known methods or process models. Unexpected events are common. Plans and software development methods and consequently weak resources for informing behavior, so people improvise.*

He also challenges lifecycle process models and instead favors teleological models, where agents choose actions to achieve goals. In line with his proposal, our findings also broadly suggest taking an empirical approach to software development. Hence, there is a need for feedback mechanisms during development, which validate an implementation against stakeholder preferences. The agile manifesto states *"We are uncovering better ways of developing software"*. Better here could be interpreted as truer, in the sense more valid; better satisfying stakeholder needs. How to conduct validation remains unanswered, however. Consequently, it is also unclear when software development is successful, and what software engineering should take as success criteria.

1.6.3 Interdependent Solution Parts

For achieving system-level feedback, our findings indicate dependencies between organizational and technical solutions. Practices within an agile way-of-working – for example the practices in Scrum or XP – both organizational and technical, may be tightly interconnected. For example, for a team to complete the backlog content for a sprint (and to populate a sprint backlog in the first place) requires dividing the implementation in small stories. Delivering work in these increments requires a platform into which change deltas can be integrated.

Sustaining confidence that system quality does not degrade over increments requires testing, likely automated (unit) tests. Success thus depends on the parts of the solution working in concert. This implies a challenge for agile adoption, that the anticipated benefits might only come about if all parts are in place; if an agile way-of-working is completely realized. This also poses a challenge for research on the fruitfulness of agile approaches. One the one hand, successful outcomes are not a sensible criteria for evaluating agile approaches; it cannot be agile only when it works. One the other hand, even an approach difficulty to adopt may ultimately turn out to be worthwhile; although it may be hard to find representative cases to study.

Similar interdependencies are present also in our findings. For example, in the principles we propose for continuous architecture evaluation, managing scope by evaluating incrementally depends on integrating the evaluation outcomes with a representation of the entire product. In the strategies our case companies are considering for scaling agile ways-of-working, product development through shared platform rather than project-specific work, the branch strategies used to achieve platform stability, and the division of responsibility by feature rather than physical component, all reinforce each other. Implementing products as a platform with a shared main branch could also help evaluation by clarify the architecture of the full integrated product.

1.6.4 Architecting

An architecture framework, as studied in Paper A, approaches architecting from a top-down perspective. Our further studies, however, shows the need for evolving the problem statement bottom-up. Nevertheless, different architecture views and knowing stakeholder concerns are useful regardless of whether evolving an architecture top-down or bottom-up. For changes where the impact becomes noticeable only after integration, for example API changes, frequent integration becomes desirable, irrespective of the overall development process used. Prior literature offers some alternative perspectives to up-front architecting. although how to evolve an architecture based on insights during implementation – in other words feedback from implementation to architecture – remains largely unexplored. Woods proposes a decision-centric view on architecture [Woo19], where architecture is a stream of decisions created as-needed, rather than a one-time activity. The architecture of a system is then a set of key decisions about that system, rather than an extensive model produced at a single point in time. This view aligns well with our proposal to articulate architecture decisions for evaluation. Waterman, Noble, and Allan [WNA15] provide a grounded theory of agile architecture that describing how agile teams decide how much up-front architecture is sufficient. Their model consists of five strategies – for example preparing the architecture for modification – that a team may choose, dependent on the context of the team and the system being built; context being characterized by six forces – for example how unstable the requirements are, and the presence of technical risk – that a team must consider when designing an agile architecture. The *Twin Peaks model* [Nus01] elaborates on alternating refinement between requirements and architecture. The model proposes developing progressively more detailed requirements and architectural specifications concurrently, which is intended to create architectural stability also when requirements change. The model does not, however, cover feedback towards the initial objectives; that is, how to evolve the system goals. For this, Kasauli et al. [KKNK17] suggest that continuous requirements engineering approaches, in particular adding value in each development iteration, may bridge the distance between implementation and system goals.

1.7 Concluding Remarks

As the centrality of software increases in automotive systems, the quick cycles possible in development of pure software systems become an alluring possibility also in this mechatronic domain. Based on practitioner perspectives, our results suggest approaches for leveraging the software aspect of automotive systems during their development.

We investigate how to achieve system-level feedback capabilities in development of automotive systems. Paper A proposes an instrument for reasoning about CI at the architecture level. We find that the concerns framed by this instrument need to be addressed during the entire development; that handling these only at the start of development is insufficient. Among these concerns are quickly responding to change. Paper B shows that current ways of working with requirements are insufficient for quickly responding to change, and suggests ways forward. Paper C investigates making feedback on an architecture manageable in an agile setting, hence responding to change on architecture. Lastly, Paper D looks broadly on responsiveness in terms of feedback-cycles beyond individual teams. Together, the findings indicate that to achieve feedback capabilities on the system-level, there is a need for tools and methods allowing artifacts on higher levels of abstraction, such as architecture descriptions and requirements, to be modified and evolve over the entire course of development.

This thesis makes three kinds of knowledge contributions regarding continuously evolving automotive systems.

1. What happens during software development for such systems.
2. How to develop software for such systems.
3. Perspectives on architecture and architecting for such systems.

Knowledge contributions 2 and 3 also serve as contributions to practice, in that they could be used for guidance in circumstances comparable to our case.

1.7.1 Future Work

Considered together, the future research directions suggested in the included papers indicate an overarching challenge. The strategies proposed for establishing system-level feedback cycles all entail organizational change. How to successfully introduce these in an organization is thus a thus a topic for change management. Changing long-established ways-of-working can in itself be difficult, and the approaches seeking to leverage the specifics of software also need to contend with approaches – for example stage-gate processes – that continue to work well for mechanical development in isolation. Achieving the quick cycle possible in software development also in system development with a mix of software, hardware, and mechanical parts is thus still a challenge for further research. As treated in the discussion, to the extent agility is the approach, further research is also needed on how to assess to which degree agility has been achieved.

Also touched on in the discussion is the need to understand the architecture of a system in terms of its implementation. In particular, more research is needed on how to bring insights from implementation back to the system level.

Lastly, in light of the criticism that software is constructed through collaboration between developers and stakeholders, rather than by developers executing a set plan, the field is open for new ways of validating software; that is, ways to assess whether developed software meets stakeholder goals. Such validation shares similarities with validation in empirical research. Consequently, one direction for future research could be empirical methods as an approach to software engineering itself.