On Interfaces to Support Agile Architecting in Automotive: An Exploratory Case Study

Rebekka Wohlrab*†, Patrizio Pelliccione*‡, Eric Knauss*, Rogardt Heldal*§
*Chalmers | University of Gothenburg, Gothenburg, Sweden
†Systemite AB, Gothenburg, Sweden
‡University of L'Aquila, L'Aquila, Italy
§Western Norway University of Applied Sciences, Bergen, Norway
Email: wohlrab@chalmers.se, patrizio.pelliccione@gu.se, eric.knauss@cse.gu.se, heldal@chalmers.se

***Abstract*—Practitioners struggle with creating and evolving an architecture when developing complex and safety-critical systems in large-scale agile contexts. A key issue is the trade-off between upfront planning and flexibility to embrace change. In particular, the coordination of interfaces is an important challenge, as interfaces determine and regulate the exchange of information between components, subsystems, and systems, which are often developed by multiple teams. In a fast-changing environment, boundary objects between teams can provide the sufficient stability to align software or systems, while maintaining a sufficient degree of autonomy. However, a better understanding of interfaces as boundary objects is needed to give practical guidance. This paper presents an exploratory case study with an automotive OEM to identify characteristics of different interfaces, from non-critical interfaces that can be changed frequently and quickly, to those that are critical and require more stability and a rigorous change process. We identify what dimensions impact how interfaces are changed, what categories of interfaces exist along these dimensions, and how categories of interfaces change over time. We conclude with suggestions for practices to manage the different categories of interfaces in large-scale agile development.**

***Keywords*-large-scale agile development; agile architecture; interfaces; boundary objects; case study; architectural change; automotive; empirical software engineering**

I. Introduction

The interaction of software and systems architecture with agile development has received increased attention in the last decades [1]. A core challenge with architecture in agile development is to deal with the tension between upfront architectural planning and agile development embracing change [1], [2], [3]. Change is an important aspect, as the rate of change impacts the adoption of the architecture-agility combination [1]. In architecture-centric agile environments, developing the architecture in a lightweight way and keeping the architecture flexible to change is challenging but necessary, especially with respect to *interfaces* [4]. Interfaces determine and regulate how two or more entities exchange information. Oftentimes, the entities connected by an interface are developed by different teams [5]. It commonly happens that blocking issues arise along interfaces among agile teams, especially if they are ambiguously specified or based on wrong assumptions [6]. To enable teams to work in an agile way, "islands of stability" are needed [7], which can be facilitated by the creation of boundary objects that maintain a common identity [8], [9]. In this context, better ways of "supporting the coordination of interfaces" are required [9].

To address this issue, mechanisms are needed to understand what architectural entities can be flexibly changed and for which parts stability is needed, especially as the complexity of systems grows [3]. Focusing on interfaces as key architecture entities, this paper contributes to an understanding of islands of stability in agile development, which interfaces can or should be particularly stable, and how they can be coordinated.

This paper presents an exploratory case study to analyze categories of interfaces and dimensions influencing their change in practice. We conducted a case study together with an automotive OEM that aims to improve the agility of their architecture practices. We present dimensions of interfaces, categories of interfaces, and suggestions for industrial practices to manage interfaces in an agile development context.

Our research questions are as follows:

RQ1: What dimensions impact how interfaces in agile automotive contexts are changed and how are the dimensions related?

RQ2: What categories of interfaces exist in the context of agile systems engineering with respect to the dimensions?

RQ3: To what degree does an interface's category change over time in agile automotive contexts?

To get a better understanding of the problem in our case study, we conducted an initial literature review and four pre-study meetings with architects at the case company. In order to collect better insights and derive guidelines, we explored the topic further using 12 semi-structured interviews with stakeholders with different roles. Our findings indicate that multiple dimensions impact how interfaces are changed, e.g., the number of affected components and the criticality. We found three relevant categories along the identified dimensions: (i) commodity interfaces, (ii) early stage interfaces, and (iii) central vehicle interfaces. An interface's category

can change over time, but typically becomes used in more components, slower to change, and less frequently changed the longer it exists. In the discussion, we provide suggestions on how to manage interfaces. We propose to leave stable and flexibly changeable interfaces up to the teams, but to control change for central vehicle interfaces. For all categories, we advise to establish interfaces to create boundaries that enable independent development, set an interface's abstraction level as high as possible, and to assess interfaces early on.

The remainder of this paper is structured as follows: In Section II, we describe related work. Section III introduces the case company and Section IV presents the research method. We present our findings related to the dimensions of interfaces (Section V), related to categories of interfaces (Section VI), and related to how categories change over time (Section VII). In Section VIII, we conclude this paper with a discussion of our findings and suggestions of practices to manage interfaces.

II. Related Work

The combination of architecture and agile development has been studied from different angles [1], with several reported challenges, practices, and lessons learned. There is still a lack of guidance of using agile practices with architecture, and the evolution and continuous change of an architecture merit further research [10], [1]. In this paper, we focus on how architecture is stabilized and changed, based on influences of product, process, and people in a large-scale agile context. Nord et al. [7] have identified the need for the "creation of islands of stability in which teams can operate in a mode that is closer to the agile sweet spot, and possibly at a faster iteration rhythm." At the same time, one aims to enable teams to respond to change quickly and capture aspects of emergent architecture [3].

This paper relates to the allocation viewtype at it is concerned with how tasks are assigned to groups in the development organization and influence both process and people aspects [11]. Concretely, this paper deals with interfaces, which are natural boundaries between agile teams developing parts of a software or system. An interface "is a form of relation between two or more distinct entities [...] such that it [...] selectively allows a transmission or communication of force or information from one entity to the other" [12]. An alternative definition is that an "interface is a boundary across which two independent entities meet and interact or communicate with each other" [11]. The notion of interface change and evolution has already been included in early works on interfaces [13], pointing to the need of leaving room for future expansion and considering design decisions that are likely to change in the future.

Interfaces capture the relation of a providing entity to a consuming entity and ideally, both are considered during the establishment and maintenance of the interface. In this study, we focus on how they are changed, from their creation until an eventual deprecation or removal. In the context of APIs, the focus often lies on the side providing interfaces that are used downstream [14]. The consuming entities are not necessarily known; instead, the focus lies on attracting more potential users of the interface as a business strategy. While APIs are mostly written for developers using the interfaces, interfaces in general can have stakeholders of several roles and can be documented using several notations [15].

III. Case Company

The case company is an automotive OEM located in Sweden. The company has had a focus on agile architecture for more than four years. The SAFe framework [6] is used widely in research and development areas across the company, across around 50 release trains with approximately 500 teams in total. Large parts of the architecture work have been moved from centralized teams of architects to the release trains and solution trains. Architects operate as a shared service in the organization. Moreover, local train architects take care of the architecture developed in particular release trains.

Parts of our interviewees work on the core system platform, representing a new, centralized vehicle architecture. The idea is that the vehicle architecture is not based on hundreds of Engine Control Units (ECUs) communicating with each other, but that a central platform is used, taking care of the communication to other ECUs via a device proxy. The core system platform is currently still under development. It will contain the most critical parts of the software, will facilitate continuous integration and deployment, and be controlled by the OEM. Microservices on a high level are used to increase decoupling between hardware and software components. An example is a microservice sending out information about the vehicle speed and it can be processed by hundreds of applications for a variety of purposes.

IV. Research Method

As our research questions are mainly of exploratory nature, we opted for an exploratory case study [16].

A. Study Design and Planning

We conducted an initial informal literature review about interfaces in software and systems architecture to understand the state of the art. Additionally, we conducted 2 pre-study workshops with 5 researchers and 2 architects, additionally 2 meetings on-site at our case company. We exploited the pre-study to understand challenges and practices and developed the initial hypothesis. Based on this understanding, we conducted 12 semi-structured interviews to analyze types of interfaces and characteristics influencing how they are managed. We created an interview guide[1] with open-ended and closed questions. After questions about the background, the guide included a presentation of our initial hypothesis

[1] https://tinyurl.com/interface-interview-guide

Table I
OVERVIEW OF THE PARTICIPANTS OF OUR CASE STUDY

No.	Role	Experience with systems/software engineering (experience in automotive)	Area of Expertise
1	Senior Software Engineer	11 yrs. (4.5 yrs.)	Central Electronic Module
2	Senior System Design Engineer	>20 yrs. (10 yrs.)	Hybrid Control System
3	Principle Engineer	13 yrs. (13 yrs.)	Active Safety and Driver Support Technologies
4	System Design Engineer	4 yrs. (4 yrs.)	Active Safety
5	System Responsible	11 yrs. (11 yrs.)	Driver Interaction and Infotainment
6	Senior Service Architect	20 yrs. (20 yrs.)	Connected Car IT Services
7	System Responsible	13 yrs. (13 yrs.)	Security Body Electronics
8	Technical Lead	20 yrs. (<1 yr.)	Application API
9	System Architect	18 yrs. (>1 yr.)	Core System Platform
10	Technical Lead	17 yrs. (<1 yr.)	Core System Platform
11	Software Dev. & Tech Lead	16 yrs. (1 yr.)	Core System Platform
12	Software Developer	7.5 yrs. (<1 yr.)	Core System Platform

that there might be different dimensions impacting interface change. Participants were asked to name examples of interfaces that they work with and characterize them.

B. Selection Criteria and Participants

After the pre-study, we selected 12 participants at two locations of the case company. We selected individuals from different departments, working with several modules/components. The participants should be knowledgeable in software or systems architecture and have different roles (e.g., developers, architects, testers).

An overview of our participants with their roles and areas can be seen in Table I. Besides indicating their experience with systems/software engineering, we also show the experience in the automotive domain.

C. Data Collection and Analysis

The lengths of the interviews were between 20 minutes and 46 minutes, with an average of 35 minutes. Afterwards, the interviewing researcher created transcripts of the interviews. We adapted Creswell's analysis approach for qualitative data to our needs [17]. We coded the interviews with an open coding approach and arrived at an initial set of 101 codes. We conducted an initial coding workshop to discuss the relations between codes after 7 interviews. We merged codes that represented the same notion and arrived at 52 codes. After performing the remaining interviews, we extended the codes by 9 additional ones. With respect to the dimensions and categories, we coded all mentioned examples as sub-codes under the codes "dimension" and "category." We arrived at the final set of categories by grouping mentioned examples of interfaces based on similar values of the dimensions. Based on the coded data, 27 examples of interfaces were given by the participants in varying levels of detail. In the interviews, we aimed to encourage participants to be as concrete as possible. We used all of the named interfaces as the basis to derive dimensions and categories. Finally, we decided how to present the findings using graphs, tables, and text.

We take this statement to explain our coding method:

> *"Internally we can change quite quickly. For safety and legal parts, it's costly validations. That's why we don't want to change a lot."*

This quote was coded with "standards and regulations" (to cover safety aspects), "testing" (as it is related to validations), "organizational boundaries" (as there is a difference between internal and external processes/interfaces), "speed of making changes", and "flexibility." The coding also allowed us to identify connections between important aspects more easily later on.

D. Threats to Validity

Our case study faces the following threats [18]:

There is a risk that we mis-heard or mis-transcribed interviewees' words. To mitigate threats to *descriptive validity*, we created transcripts that allowed us to conduct an exact analysis of what was said in the interviews.

To reduce threats to *interpretive validity*, we presented the study's context at the beginning of the interviews. We paraphrased interviewees' statements and used their own terminology to ensure correct understanding.

Theoretical validity is related to beliefs and conceptions we had prior to the study. We thoroughly discussed the data to ensure a chain of evidence. We aimed to present our method and reasoning in a transparent way.

Generalizability is about how applicable our findings are in other contexts. The goal of our exploratory study was to understand the concrete context of the case company. The categories we identified are based on examples of interfaces mentioned by individuals. We used data triangulation of interview data from several interviewees with different roles to strengthen internal generalizability. To be transparent about our particular case, we described the characteristics of the case company and participants. The participants in this study had at least four years of experience with systems/software engineering. The experience in automotive is shown in parentheses in Table I. While the automotive domain comes with particular challenges (not pure software development, high criticality, diversity of functions, etc.), the findings are not specific to automotive systems and are likely valuable also in other large-scale agile contexts.

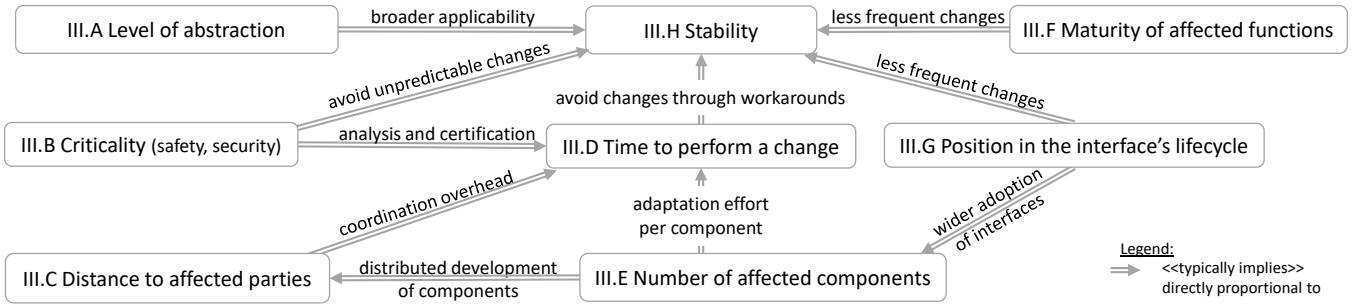

Figure 1. Dimensions of interfaces and their relations. X «typically implies» Y means that an increase in X typically implies an increase in Y, which can be due to direct conceptual relations between dimensions or because of actions performed by stakeholders.

Threats to *evaluative validity* exist as researchers potentially present findings in a judgmental way, based on their own values and understanding. By collecting feedback from fellow researchers on our interpretations and the presentation of our findings, we tried to mitigate this threat. One should note that the goal of this research was rather to explore the topic of interfaces in agile automotive contexts, rather than evaluating a method.

V. Dimensions of Interfaces

We identified several dimensions of interfaces and relationships between them. We consider dimensions as measurable characteristics of interfaces and how they are changed. In this section, we answer RQ1: *What dimensions impact how interfaces in agile automotive contexts are changed and how are the dimensions related?*

An overview of our findings is displayed in Figure 1. The arrows represent implications between dimensions and the dimensions are directly proportional to each other, i.e. the higher one dimension, the higher the other. This increase can be due to the direct conceptual relations between the dimensions (e.g., a higher criticality implies a longer time to perform a change because of additional overhead) or because of decisions taken or actions performed by stakeholders (e.g., if the time to perform a change is high, typically an interface is kept more stable by architecture stakeholders to reduce the overhead of changing it). The text on the arrows indicates the core motivator behind the relations.

The *level of abstraction* refers to the nature of an interface and how closely related to hardware it is. The *criticality* relates to safety and security criticality. The *distance* to affected parties relates to the distance between people (e.g., geographical, organizational, or cognitive distance), related to artifacts (e.g., semantic distance), and related to activities (e.g., temporal distance) [19]. The *time to perform a change* relates to the time it takes to plan and implement a change. The *number of affected components* relates to how many logical, software, or hardware components need to be adapted because of a change.

In the automotive domain, functions of different natures exist that are more or less established. We found that the *maturity of the affected functions* of an interface is relevant. Another dimension is the *interface's position in the lifecycle*. As any artifact, an interface has a lifecycle from its creation to its first use to its maintenance, its potential deprecation, and removal. Finally, the *stability* of an interface indicates how steady it is over time.

We describe each dimension in the following, together with its outgoing relations shown in Figure 1. Each dimension is marked with its subsection's letter.

A. Level of Abstraction

Interfaces at our case company were typically signals and communication interfaces between ECUs, interfaces between software components, or application programming interfaces (APIs) on a higher level of abstraction.

Signals and interfaces between ECUs are managed in a signal database. The intention with hardware-related interfaces is to stabilize them early on, as stressed by Interviewee 7. However, currently, the signal database is constantly changed. Change requests come with an overhead as a signal database team needs to approve and perform the changes.

Interfaces on a software level (e.g., APIs) are intended to be reusable over time and make the company less dependent on the implementation (e.g., provided by suppliers). Several interviewees reported that changing an API on a high level of abstraction should be avoided.

> *"With APIs, it's difficult to change or remove things, it's easy to add. If you think 'this is good for now, we will see later if it will change,' that will be problematic."* (Interviewee 12)

We identified that a higher level of abstraction typically implies a higher level of *stability*. This is mainly motivated by the *broader applicability* of a high-level interface. Interfaces on a high level of abstraction, e.g., APIs, can be generic to be usable for different applications and have the potential to be stable. Interviewee 10 stated that "*the more we abstract from unnecessary details, the more stable our interfaces can be*."

B. Criticality

Eight of our interviewees pointed out that criticality is an important factor that requires to keep interfaces *stable*. Interviewee 2 reported that "*since we have safety-critical functions in the car, we need to have stable interfaces*", and Interviewee 1 stated that "*costly validations*" need to be performed when changing safety-critical interfaces. A core motivator for the stability of safety-critical interfaces is the need to do the *certification and security/safety analysis* again. These additional steps increase the *time to perform a change*. Additionally, practitioners are afraid that a change could have *unpredictable consequences*, which is why they prefer to have stable interfaces. For this reason, a higher level of criticality typically implies more stability.

C. Distance to Affected Parties

This dimension relates to the distance between human stakeholders, semantic distance, and the distance between activities [19]. Organizational distance is relevant both within the same company and to suppliers or other companies. Another relevant aspect is whether involved teams have the mandate to prioritize tasks in a common way. Interviewee 3 stated that "*it's tougher to change interfaces [...] when you are affecting teams and components outside of your scope where you don't have control.*"

All interviewees mentioned that *performing a change takes more time* if several (distributed) teams are affected by a change. The more stakeholders are affected by a change, the higher will be the *coordination overhead*.

Interviewee 9 stated that if only two teams are affected, resolution of changes is bilateral, independent, and quick.

> *"Sometimes it's just two teams, then they talk. Then one team takes over the concern and discusses it with the other one. And [the architects] give directions."*
>
> (Interviewee 9)

D. Time to Perform a Change

Our interviewees mentioned that the speed of implementing a change also impacts how often a change can be made and how *stable* an interface is required to be kept. It clearly depends on the type of interface how long a change takes, but can be up to half a year (Interviewee 12). Because of these constraints, engineers sometimes try to come up with *workarounds to avoid making a change*. As time is related to cost, several interviewees see it as desirable to reduce the time to perform a change and increase the stability.

> *"You want to be able to change the API fast. If you have a slow process, then you get problems in agile because the teams get blocked."* (Interviewee 10)

To foster efficient development, practitioners try to keep these interfaces as stable as possible.

E. Number of Affected Components

The *number of affected components* relates to how many parts of the system need to be adjusted because of a change. Several interviewees pointed out that the number of affected components or subsystems by an interface change strongly increases the *time a change takes*, because of the *adaptation effort per component*. Interviewee 4 stated that "*[the time to change an interface] depends on the change, how many subsystems are affected and what actions need to be taken.*" It is common that these components are also managed by different teams across a *distributed organization*: a high number of affected components typically implies higher *distance to affected parties*.

F. Maturity of Affected Functions

A variety of functions are developed in automotive companies, e.g., infotainment, powertrain, or autonomous driving functions. We found that the maturity of affected functions has a strong impact on the *stability* of their interfaces, due to *less frequent changes* occurring for these functions. For instance, Interviewee 2, working with the hybrid control system in the powertrain department, stated that they "*mainly have stable interfaces overall.*" On the other hand, Interviewees 8–12, working within new functions, stated that the frequency of change is very high and the interfaces are very unstable.

> *"We have different parts and different features of the car with different maturity periods. Today a lot of functionality is quite mature. [...] So we don't need to change those interfaces. Those are related to traditional car functions. The commodity functionality, what the customer expects."* (Interviewee 1)

Several interviewees stated that the surrounding layers of functionality impact the number of interface changes:

> *"After some time, when the layers that are using your interface are not really subject to change, then you see that it is stable."* (Interviewee 10)

All interviewees stated developing new, immature functionality, the frequency of change is higher.

> *"In the early phases, it is more unclear and technology decisions will change a lot. While in the end, it's just nuances of which [interfaces] to change and how often."* (Interviewee 5)

G. Position in the Interface's Lifecycle

An interface can be at different points in its lifecycle, from its creation to its deprecation or removal. The later it is in its lifecycle, the more *stable* it becomes. Interviewee 11 stated that an interface undergoes "*many changes [at first], then some rework, and then hopefully it will stabilize over time.*" Interviewees 6 and 7 stated that the period of change is typically two to three years. Then, the stability increases due to *less frequent changes*.

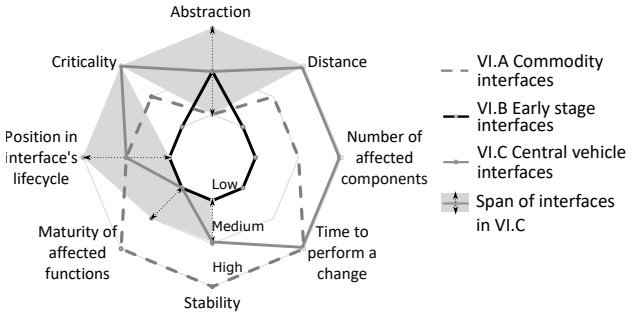

Figure 2. Categories of interfaces with their dimensions

Our interviewees pointed out that the longer an interface exists, the *more components* are likely to use it and rely on it. An interface gets *more widely adopted* with time. As a consequence, the time of making changes and the stability as a whole increase.

> *"Before everyone else uses it, [...] update it daily, every hour, I don't care. But when you work with people who use the API, you need handshakes between the provider and users."* (Interviewee 10)

Interviewee 4 noted that when an interface is included in production vehicles' software, it takes even more time to change it.

H. Stability

Many of the mentioned dimensions impact the stability of interfaces. Stability is understood as the opposite of the frequency of change. The stability of an interface can be a consequence of the development context or actively enforced by stakeholders. Our interviewees use different formulations to talk about stability. It can be observations of changes, e.g., "*it becomes rather stable*" (Interviewee 6). However, practitioners also intervene as there is a "*need to stabilize [an interface]*" (Interviewee 9). Interviewee 2 stressed that "*we want to have stable interfaces, then we can make our own internal development.*" Interviewee 5 pointed out that there exist interfaces that are naturally stable (e.g., because the affected components do hardly ever change), as well as interfaces that are kept stable in an artificial way ("*because [the process to change] is so rigid*" or because suppliers are involved).

VI. Categories of Interfaces

This section answers RQ2: *What categories of interfaces exist in the context of agile systems engineering with respect to the dimensions?*

We used the interview data to scrutinize examples of interfaces and arrived at the following categories: (i) commodity interfaces, related to mature functions, (ii) early stage interfaces, used in few components and being changed quickly and often, and (iii) central vehicle interfaces, which are very critical interfaces that are used across a large distance and affect many components.

Interfaces can be at different positions in their lifecycle and all dimensions can change over time. According to our findings, one interface can only belong to one of these categories, although it can change its category with time.

Table II depicts the categories with their dimensions, as well as main concerns with each of the categories. The categories with their dimensions are shown in Figure 2 as a radar chart. The dimensions are shown along the axes, with "Low", "Medium", and "High" as the dimension values. The dashed gray line indicates the values for commodity interfaces (Section VI-A), the black solid line for early stage interfaces (Section VI-B), and the gray solid line for central vehicle interfaces (Section VI-C). The category of central vehicle interfaces consists of interfaces with slightly different characteristics. The gray area in the figure indicates the span of central vehicle interfaces. Central vehicle interfaces vary along the lifecycle, abstraction, maturity, and stability axes. The position in the lifecycle varies naturally as time passes. While the abstraction and maturity impact the (potential) stability of an interface, the main influencing property of central vehicle interfaces is its centrality, as many components are affected. We describe subcategories within this category, but group them as central vehicle interfaces.

The time to perform a change in the identified categories is either Low or High. The values are based on the gathered data and perceptions of interviewees. Future work is needed to accurately measure the time to perform a change for different interface types.

A. Commodity Interfaces

Commodity interfaces relate to *very mature* functions that have been developed and used for several years. Interviewee 7 mentioned signals for the headlamps technology on a rather low level of *abstraction*. An interface needed to be adjusted due to minor changes in the technology, but implementing the change took a high *time effort*. Interviewee 7 stated that the *stability* is high now, and that the interface affected a medium number of *components* and only organizational units across a manageable *distance*. The *criticality* of the interface is neither particularly high nor low, but undergoes the same procedures as typical vehicle interfaces.

Interviewee 2 named interfaces for gearbox types as an example of a commodity interface that had to be adjusted because a new gearbox type was introduced. The team has a strong focus on actuators and controllers for them, on a rather low level of *abstraction*. Only groups inside the same department are affected by the change, across a *medium distance*. They develop 3-4 nodes, i.e., a medium *number of affected components*. The interface is *stable* now, as Interviewee 2 mentioned: *"We came up with a common interface that could take care of both gearbox types. Like a*

Table II
OVERVIEW OF INTERFACE CATEGORIES AND THEIR DIMENSIONS

	Abstraction	Distance	Number of components	Maturity of functions	Position in interface's lifecycle	Criticality	Time to change	Stability	Main concerns
VI-A: Commodity	Low	Medium	Medium	High	Medium	Medium	Long	High	Prioritization of interface changes
VI-B: Early stage	Medium	Low	Low	Low	Early	Low	Short	Low	Rework due to lack of stability
VI-C: Central vehicle	Medium	High	High	Low	Medium	High	Long	Medium	Lack of flexibility, high impact of change
VI-C1: Critical cross-company	Medium	High	High	Medium	Late	High	Long	Medium	
VI-C2: Service API	High	High	High	Low	Early	High	Long	Low	
VI-C3: Infotainment head unit	Low	High	High	Low	Medium	High	Long	Medium	

super interface. That made it quite stable. And we strive for stable interfaces." However, it took a year of simulations to assess the consequences of the change and establish it (i.e., a long *time* to perform a change). Our interviewees stated that differently prioritized backlogs are an issue when performing a change.

> *"We always struggle with having the resources and prioritizing it the same way. We can't get resources to do the work we find most challenging."*
>
> (Interviewee 2)

B. Early Stage Interfaces

Early stage interfaces are fundamentally different from commodity interfaces. Interviewee 8 mentioned an interface between *two components*: the vehicle interface unit that is connected to an ECU, and the core platform itself. It is on a medium *level of abstraction*, neither on a high API or service level, nor on a signal level. Interviewee 8 stated that only two teams are affected who sit in the same building: *"They can do whatever they want. [...] Because they own both the software running here and the software that receives the data."*

The *time* to perform changes is short, as the teams can change interfaces "*without disturbing anyone else*" (Interviewee 8). The *criticality* is considered low in the current development context, although our interviewees mentioned the need to consider criticality in the future. The *stability* is currently low, as many decisions are being explored and changed. Our interviewees stated that higher stability would be beneficial.

> *"We need more upfront work, it's too little right now. [...] The penalty is that they need to rework sometimes."*
>
> (Interviewee 9)

C. Central Vehicle Interfaces

Central vehicle interfaces connect a high number of critical components developed across large distances, and take a long time to change. The main concern with central vehicle interfaces is the lack of flexibility and the high impact of change. Due to the high distance and the high criticality, changes have to be performed in a very controlled way. To increase the interfaces' stability, the trend is to increase the level of abstraction and support a larger number of applications of the interface. However, Interviewee 6 pointed out that the interfaces still "*need to be revised all the time because the different teams [...] find a lot of details that nobody thought about.*"

We found three subcategories that we describe in the following: critical Cross-company interfaces, a service API, and infotainment head unit interfaces.

1) Critical Cross-Company Interfaces: Interviewee 6 mentioned interfaces related to connectivity, both inside of our case company, but also to other organizations. The *distance* is high: 17 architects are involved in-house to manage these interfaces, plus additional ones from other companies. Due to the large number of stakeholders, the *time* to agree on and perform changes is long. The *number of affected components* is high, which is why the intention is to keep the interfaces as *abstract* as possible, so that they can be used for different purposes and applications. The interfaces are highly *critical*, both as they provide safety features and because of security and privacy constraints. The *stability* is on a medium level, but typically increases over time: "*Usually, in the first three years it changes a lot and then it becomes rather stable.*" The interface is at a *late position* in its lifecycle, with only 6 months before production. The affected functions are on a medium level of *maturity*, i.e., neither very early nor very established.

Interfaces are especially complicated if they are used to exchange data with externals:

> *"For interfaces to other companies, you try to keep them as minimal as possible. [...] We try to focus not only on the technical interfaces, but more on the data structure as such. So that we are not bound to a certain technology, it's still changing so fast."*
>
> (Interviewee 6)

2) Service API: A service API has been mentioned by Interviewees 8–12 and represents interfaces to microservices. Software service interfaces are on a *high level of abstraction*. Interviewee 10 pointed out that they are security-*critical*, as "*not everyone should have access to all signals*." Interviewee 8 mentioned that some interfaces affect a high *number of components*: *"If we make one change, maybe we need to change 100 applications"*.

There is a *large distance* to affected parties, both because of different backgrounds and different geolocations. According to Interviewee 8, the *time to perform a change* is high, due to a high overhead of interface governance. So far, the *maturity of affected functions* is very low. Also the *position in the interface's lifecycle* is in its definition phase. Interviewee 12 stated that eventually, it will become very hard to change and to remove interfaces in the API. It "*should be really stable*" (Interviewee 8), because of the large impact on other teams. Currently, however, it has a low *level of stability*.

3) Infotainment Head Unit Interfaces: Interviewee 5 mentioned interfaces to and from the central head unit within infotainment. These interfaces allow users to activate and use functions in the car. It has a high *number of affected components* and also affects different departments across *large distances*.

Interfaces are often on a signal level (i.e., low *level of abstraction*). The *criticality is high*, and certain signals need to be prioritized more than others, as Interviewee 5 stated: "*We can delay features like a shuffle function. But warnings for the safety system are more important, for instance.*" Changing the database takes a long *time*, as all changes are performed by a signal database team. It is a concern, as *"it's not really scalable if you want to change your system rather often"* (Interviewee 4).

The *maturity of affected functions* is low, as infotainment needs to adapt quickly to market trends. The interfaces' *position in the lifecycle* is currently under development, but has passed the initial definition phase. The *stability* is currently on a medium level, but the interfaces still undergo change.

VII. Change of Categories over Time

This section answers RQ3: *To what degree does an interface's category change over time in agile automotive contexts?*

Our overall observation is that an interface's category can change over time, but typically the values of the dimensions increase rather than decrease. The position in the interface's lifecycle inevitably changes over time and moves from "Early" to "Late." Our interviewees named interfaces across all lifecycle positions. An increase in the position in the interface's lifecycle affects three other dimensions (Section V-G): More components are affected, more time is needed to perform a change, and stability is increased.

Early stage interface, in particular, change values in several dimensions:

> *"At one point in time, you should try to be careful with interface changes. [...] We have a year of a lot of changes, but then we want a release so that others can rely on this."* (Interviewee 9)

Commodity interfaces are already at a stable level and our interviewees did not expect their dimensions to change, except if new, immature functions affect the interface. Interviewee 2 reported that interface changes are motivated by "*new technology or new functions that put requirements on us*."

For the central vehicle interfaces, the criticality and distance complicate change. Several interviewees mentioned a "*point of increased stability*" that will be reached with an increasing use of the interfaces. Interviewee 12 motivated this as follows: "*Once it comes to a point where people will use it, other people, other groups, then it won't be easy to change it anymore*".

The goal is to aim for stability in certain time periods:

> *"We don't change constantly but have a heartbeat. So that the API should stay as stable as possible between two points in time."* (Interviewee 10)

The interviewee recommended to keep the API stable and continue the development on a separate branch that is merged later on. Interviewee 10's idea is to start with "*one or two users at the beginning. [...] With a small set of users, it is easier to get the standard in place.*".

VIII. Discussion and Conclusion

This paper describes dimensions of interfaces that impact change, categories along the dimensions, and an analysis of how these categories change over time.

We conclude that interfaces play a central role for the architecture, especially when developing multiple components that exchange information. Related work confirms that challenges with agile development are to foster information exchange across teams and to deal with distance between teams [19], [20], [21]. Our findings indicate that practitioners aim for stable interfaces to enable flexible and independent development inside agile teams [7]. In this context, architecture models and descriptions have been considered boundary objects [9], [8], i.e., artifacts establishing a common understanding across sites, while allowing each of the teams to maintain its interpretation of the object [22]. Our understanding is that interfaces as an integral part of the architecture can also be considered boundary objects between teams developing different parts of the system. Teams interpret them from different angles, based on the concerns of the component they are currently working on. Architects have a broader scope and work with interface or architecture descriptions to communicate with several teams [15]. The findings presented in this paper help to understand the

Table III
SUGGESTIONS FOR PRACTICES AND RELATIONS TO DIMENSIONS

	Practice	Dimensions
1	Select what interfaces need to be managed centrally, and leave other interfaces (especially those in an early stage) up to the teams.	Time, position in lifecycle
2	As soon as the lack of stability impedes that teams can work autonomously, involve stakeholders with architecture expertise to establish interfaces that set the boundaries.	Stability
3	Increase an interface's level of abstraction if you have the flexibility, so that you can support more users.	Abstraction
4	Assess interfaces in the early stages with one or two users before its position in the lifecycle and stability increase.	Position in lifecycle, stability
5	For central vehicle interfaces, establish controlled change mechanisms and a strategy for versioning.	Number of components, stability

different dimensions relevant for managing interface change and assessing these boundary objects' relevance and usage over time [8].

Based on the mentioned examples in the study, we identified three categories of interfaces with respect to dimensions and main concerns:

- Commodity interfaces, related to mature functions. They only affect teams across a manageable distance, but changes take time to prioritize, assess, and agree upon.
- Early stage interfaces are in an exploration phase and can be changed flexibly, although too frequent changes result in undesired rework.
- Central vehicle interfaces, that play a role for many architectural entities. They are more difficult to manage than the former two categories, being very critical and affecting many components and teams.

Table III shows suggestions for practices that we deduced based on the findings and how they relate to the dimensions. The practices are based on our interviewees' experiences, their suggestions, and reasoning based on the collected data. In the following, we motivate the suggestions based on a re-elaboration of our findings.

First practice: *Select what interfaces need to be managed centrally, and leave other interfaces (especially those in an early stage) up to the teams.* Currently, the tendency at our case company is to control interfaces from a central point, involving architects or the signal database team. This results in a large overhead of managing interfaces and long *times to perform a change*. To be scalable and competitive in the future, a more lightweight approach can be beneficial to manage early stage interfaces impacting only a few teams. Interviewee 8 mentioned that should be "*up to the teams to decide [on these interfaces].*"

Second practice: *As soon as the lack of stability impedes that teams can work autonomously, involve stakeholders with architecture expertise to establish interfaces that set the boundaries.* Several interviewees suggested that if an interface concerns only one or two teams, they can agree on changes themselves, keeping the architects in the loop. The suggestion is also confirmed by related work [6]. As *lack of stability* often results in rework (Section VI-B), boundaries as "islands of stability" [7] need to be set.

Third practice: *Increase an interface's level of abstraction if you have the flexibility, so that you can support more users.* More general and *abstract* interfaces often lead to higher stability [13]. In the case of interfaces for gearbox types, a more abstract "super interface" was created to accommodate two types of gearbox interfaces (Section VI-A). A similar observation was made in the cases of central vehicle interfaces and the abstraction from details.

Fourth practice: *Assess interfaces in the early stages with one or two users before its position in the lifecycle and stability increase.* When examining the change of categories over time (Section VII), we found that flexible interfaces likely reach a *point where change becomes more complicated*. Our interviewees' idea was to leverage the freedom in initial phases and assess interfaces with a few users before more *stability* is required.

Fifth practice: *For central vehicle interfaces, establish controlled change mechanisms and a strategy for versioning.* Central vehicle interfaces that have become used by *many components* should be carefully changed (Section VI-C). Also in agile setups, *stability* is needed to enable teams' autonomy, as confirmed by [7].

To conclude, the study presented in this paper contributes to an understanding of architectural change in agile contexts. The described dimensions of interfaces and their relations can help to understand characteristics of interfaces and how they can be managed in agile development. While certain categories can be managed in a very flexible way, other categories of interfaces should be stabilized for agile teams to work effectively. The presented categorization showed that the dimensions are useful to describe differences between interfaces. We observed that over time, dimensions rather increase than decrease.

It should be noted that the extracted categories are based on an initial exploratory case study and that more research is needed to systematically examine categories of interfaces. We expect that there are more general categories of interfaces that are not specific to the automotive domain.

Practitioners can understand dimensions and categories of interfaces to reflect on how interfaces can be established as "islands of stability" to enable agile teams to flexibly develop systems and software [7]. Our findings suggest that interfaces with different characteristics require different approaches to managing change. The suggestions for practices can help as a starting point to create and change interfaces that allow as much stability as needed.

The study was conducted in the automotive domain,

characterized by very diverse functions, criticality, slow processes for non-software parts, and large distances between stakeholders. We expect these findings to be valuable also for other companies and industries, but further research is needed. Based on this study and the suggested practices, researchers can assess interfaces in other companies and domains, and develop tools and methods to support interface change in agile environments.

ACKNOWLEDGMENTS

We would like to thank all anonymous participants and facilitators for the time and effort in supporting this study. Moreover, this study was supported by the Vinnova projects NGEA and NGEA step 2, the Software Center Project #27 (software-center.se), and the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

REFERENCES

[1] C. Yang, P. Liang, and P. Avgeriou, "A systematic mapping study on the combination of software architecture and agile development," *Journal of Systems and Software*, vol. 111, pp. 157 – 184, 2016.

[2] P. Pelliccione, E. Knauss, R. Heldal, S. M. Ågren, P. Mallozzi, A. Alminger, and D. Borgentun, "Automotive architecture framework: The experience of Volvo Cars," *Journal of Systems Architecture*, vol. 77, pp. 83 – 100, 2017.

[3] M. Waterman, J. Noble, and G. Allan, "How much up-front? a grounded theory of agile architecture," in *Proceedings of the International Conference on Software Engineering (ICSE '15)*, 2015, pp. 347–357.

[4] K. Read and F. Maurer, "Issues in scaling agile using an architecture-centric approach: A tool-based solution," in *Proceedings of the 3rd XP Agile Universe Conference*, 2003, pp. 142–150.

[5] K. Rahmani and V. Thomson, "Managing subsystem interfaces of complex products," *International Journal of Product Lifecycle Management*, vol. 5, no. 1, p. 73, 2011.

[6] D. Leffingwell, *Scaling Software Agility: Best Practices for Large Enterprises (The Agile Software Development Series)*. Addison-Wesley Professional, 2007.

[7] R. L. Nord, I. Ozkaya, and P. Kruchten, "Agile in distress: Architecture to the rescue," in *Proceedings of the 15th International Conference on Agile Software Development (XP 2014)*. Springer International Publishing, 2014, pp. 43–57.

[8] R. Wohlrab, P. Pelliccione, E. Knauss, and M. Larsson, "Boundary objects in agile practices: Continuous management of systems engineering artifacts in the automotive domain," in *Proceedings of the 2018 International Conference on Software and System Process (ICSSP '18)*. ACM, 2018, pp. 31–40.

[9] L. Pareto, P. Eriksson, and S. Ehnebom, "Architectural descriptions as boundary objects," in *Proceedings of the 13th International Conference on Model Driven Engineering Languages and Systems (MODELS 2010)*, 2010, pp. 406–419.

[10] T. Dingsøyr, N. B. Moe, T. E. Faegri, and E. A. Seim, "Exploring software development at the very large-scale: a revelatory case study and research agenda for agile method adaptation," *Empirical Software Engineering*, vol. 23, no. 1, pp. 490–520, Feb 2018.

[11] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, R. Nord, and J. Stafford, *Documenting Software Architectures: Views and Beyond*, ser. SEI series in software engineering. Pearson Education, 2003.

[12] B. Hookway, *Interface*, ser. Cultural studies. MIT Press, 2014.

[13] D. Hoffman, "On criteria for module interfaces," *IEEE Transactions on Software Engineering*, vol. 16, no. 5, pp. 537–542, May 1990.

[14] J. Lindman, J. Horkoff, I. Hammouda, and E. Knauss, "Emerging perspectives of API strategy," *IEEE Software*, pp. 1–1, 2018.

[15] F. Bachmann, L. Bass, P. Clements, D. Garlan, J. Ivers, R. Little, R. Nord, and J. Stafford, "Documenting software architecture: Documenting interfaces," CarnegieMellon Software Engineering Institute, Tech. Rep., 2002.

[16] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Software Engineering*, vol. 14, no. 2, p. 131, Dec 2008.

[17] J. W. Creswell, *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches*, 3rd ed. Sage Publications Ltd., 2008.

[18] J. Maxwell, "Understanding and validity in qualitative research," *Harvard Educational Review*, vol. 62, no. 3, pp. 279–301, 1992.

[19] E. Bjarnason and H. Sharp, "The role of distances in requirements communication: a case study," *Requirements Engineering*, vol. 22, no. 1, pp. 1–26, Mar 2017.

[20] K. Petersen and C. Wohlin, "The effect of moving from a plan-driven to an incremental software development approach with agile practices: An industrial case study," *Empirical Software Engineering*, vol. 15, no. 6, pp. 654–693, 2010.

[21] M. Hummel, C. Rosenkranz, and R. Holten, "The role of communication in agile systems development: An analysis of the state of the art," *Business and Information Systems Engineering*, vol. 5, no. 5, pp. 343–355, 2013.

[22] S. L. Star and J. R. Griesemer, "Institutional ecology, 'translations' and boundary objects: Amateurs and professionals in Berkeley's museum of vertebrate zoology, 1907-39," *Social Studies of Science*, vol. 19, no. 3, pp. 387–420, 1989.