Identifying and Visualizing Architectural Debt and Its Efficiency Interest in the Automotive Domain: A Case Study

CONFERENCE PAPER · SEPTEMBER 2015

READS
291

4 AUTHORS, INCLUDING:

Ulf Eliasson
Chalmers University of Technology
2 PUBLICATIONS **0** CITATIONS
SEE PROFILE

Antonio Martini
Chalmers University of Technology
13 PUBLICATIONS **31** CITATIONS
SEE PROFILE

Identifying and Visualizing Architectural Debt and Its Efficiency Interest in the Automotive Domain: A Case Study

Ulf Eliasson*, Antonio Martini†, Robert Kaufmann†, Sam Odeh†
*Volvo Car Group, Sweden
ulf.eliasson@volvocars.com
†Chalmers University of Technology | University of Gothenburg
Department of Computer Science and Engineering.
antonio.martini@chalmers.se, {robkau, osam}@student.chalmers.se

***Abstract*—Architectural Technical Debt has recently received the attention of the scientific community, as a suitable metaphor for describing sub-optimal architectural solutions having short-term benefits but causing a long-term negative impact. We study such phenomenon in the context of Volvo Car Group, where the development of modern cars includes complex systems with mechanical components, electronics and software working together in a complicated network to perform an increasing number of functions and meet the demands of many customers. This puts high requirements on having an architecture and design that can handle these demands. Therefore, it is of utmost importance to manage Architecture Technical Debt, in order to make sure that the advantages of sub-optimal solutions do not lead to the payment of a large interest. We conducted a case study at Volvo Car Group and we discovered that architectural violations in the detailed design had an impact on the efficiency of the communication between components, which is an essential quality in cars and other embedded systems. Such interest is not studied in literature, which usually focuses on the maintainability aspects of Technical Debt. To explore how this Architectural Technical Debt and its interest could be communicated to stakeholders, we developed a visual tool. We found that not only was the Architectural Debt highly interesting for the architects and other stakeholders at VCG, but the proposed visualization was useful in increasing the awareness of the impact that Architectural Technical Debt had on efficiency.**

I. Introduction

A modern vehicle, such as a car, is a complex system of mechanical components, electronics, and software. Depending on the configuration, it could contain nearly a hundred Electronic Control Units (ECU), small to PC-sized computers, connected in several networks within the car. Taking care of the architecture of such a complex system is both challenging and crucial.

At Volvo Car Group, the architecture for the electrical system is created by a system architecture group while the detailed design is left to the engineers in the different subsystems and components. No automatic transformation is done between the architecture models and the detailed design. Instead, the detailed design model of the system is stored in a proprietary custom tool. Therefore, there might exist inconsistencies between these two levels of abstraction [4]. These inconsistencies could lead to Architecture Technical Debt (ATD), which is currently hard to discover and might create several issues.

ATD is a recently coined metaphor that is used to describe sub-optimal architectural solutions that have a short-term benefit (taking the debt) but have a negative impact in the long run (paying the interest).

According to a recent mapping study [7], ATD is studied to some extent in literature, but there are still several gaps when practically managing such kind of debt: the interest (impact) often remains hidden, which makes it difficult for the stakeholders to decide if the refactoring of ATD should be performed or not. Many studies focus on the maintainability impact that an ATD item has, but not on other quality attributes, such as performance.

Another gap is that ATD is studied at a rather general level, but there is a lack of empirical studies showing what ATD is and how it is managed in different contexts, for example in the automotive industry.

Finally, there is little research about how to visualize ATD and its interest (impact), especially in a combined view.

Based on the previous gaps, we have constructed the following three research questions:

- RQ1 What is a typical ATD item for the automotive industry?
- RQ2 What is the impact, or interest, of such ATD item?
- RQ3 How can the ATD and its interest be visualized together to aid the stakeholders' awareness?

In this study, we have explored how to visualize the efficiency interest caused by architectural violations in a concrete case. We have identified, together with the architects at VCG, relevant ATD items present at the design level but forbidden at the architecture level. Then we have investigated the interest in terms of efficiency (a key quality for cars) due to the existence of such ATD items. We have constructed a visualization tool based on the literature state of the art and the ATD, and we have validated the tool and visualization with architects at the company.

In section II we describe the context of our study as well as related research. To answer our research questions, we conducted a case study outlined in section III. The results of the case study is described in IV. The paper is concluded with a discussion in section V and our conclusions and future work in VI.

II. Background

A. Architectural Technical Debt (ATD)

Initially, technical debt was focused on software implementation at code level [7], [1], but it has since been expanded to include software architecture, design, documentation, requirements, testing [1], and lately even social debt in software engineering [15]. Architectural debt is the one relevant to this study.

Architectural technical debt is considered as sub-optimal solutions with respect to an intended architecture [8]. In case of dependencies, a sub-optimal solution can be considered the presence of a dependency in the implementation that is not allowed in the architecture description (or else a violation of an architectural rule). However, in some cases, the architecture is composed of more layers, needed in order to manage large and complex systems: at VCG there exist two different layers of abstraction, a design and an architecture level. In such context, we found, ATD can also be considered as an inconsistency between the two levels, for example having dependencies at the design level that are forbidden at the architecture level.

According to the Technical Debt theoretical framework, ATD is composed of the following elements: the sub-optimal solutions (discussed previously), the cost of refactoring them (principal) and the their impact (interest) on some quality attributes, for example on maintainability or performance. Based on the information about these three components, software developers, architects and managers would be able to decide if an ATD item is a good or bad investment (for example if the interest paid is more than the cost of refactoring), in order to allocate resources for refactoring. In this study, we aim at supporting such decisions by identifying ATD items (architectural violations) in the particular context of VCG and by understanding and visualizing their impact (interest) in terms of efficiency of the system (an important non-functional requirement in the automotive domain).

B. Visualization of ATD and Its Interest

To manage the complexity of large software (and consequently large architectures), the software engineering community has developed techniques related to software visualization, to enhance the understandability of the systems [11]. As software architectures are becoming more complex and utilized in the management of large-scale software [13], architectural visualization becomes an important aspect of software visualization, helping architects and other stakeholders explore the system effectively [6].

In software visualization, software artifacts are presented to developers as graphical components, either two or three-dimensional representations, that assist with human comprehension of software and making hidden structures and relations visible [10]. As with many aspects of software engineering, the challenge with software visualization is to utilize the correct tool given a particular problem, visualizing the right thing and therefore improving software comprehension [10].

When visualizing software architectures, the two main areas to consider are the presentation of the software structure as well as the relationships between components in the system [2]. However, there are also other factors to consider when visualizing software. For instance, [10] proposes run-time behavior, metrics, and software evolution as possible candidates. In the meta-analysis by [6], these are not necessarily mutually exclusive and can be combined.

In the recent field of TD (and specifically of ATD), software visualization would be important for the stakeholders in order to understand architecture sub-optimal solutions in combination with the visualization of the other metrics related to the cost of refactoring (principal of ATD) and the interest (impact of ATD). Only by visualizing these three components together it would be possible to evaluate and prioritize the ATD items present in the system.

However, studies on how to provide such visualization, according to [7], are missing in the current body of knowledge. In this study, we aim at filling this gap by exploring the employment of a visualization technique representing the ATD items together with their interest. Such visualization has been developed using the state of the art in the software visualization field, together with the continuous input from the stakeholders (architects) at VCG.

C. Electrical Architecture at Volvo Car Group

The software architecture at VCG can be divided into three different views: the *Logical view*, detailing how groups of functionality should interact in the system; the *Design View*, detailing how the system is actually constructed with individual units of functionality; and the *Deployment*, detailing how each unit of functionality is implemented as software packages and deployed on different ECUs. The system architecture group constructs the logical view as the intended architecture. The design, however, is performed by subsystem and software owners in the different development groups. The deployment is done partly by the architecture group and partly by the subsystem and software owners.

D. Logical View

At the highest level of abstraction, there is the logical view that defines the intended architectural structure of the system. It consists of Logical Architectural Components (LAC) representing a logical group of functions. A simplified example could be an LAC responsible for locking the car. Each LAC could connect to one or more other LACs, representing communication between them and subsequently dependencies. Figure 1 depict an example of LACs communicating. The logical view exists in the form of a UML-like model authored by the system architecture group.

E. Design View

The design view determines the actual structure of the system, breaking down the LACs into smaller Logical Components (LC), each representing a single unit of functionality. If we continue our simplified example with locking, the LAC responsible for the complete locking function can be broken down into for example one LC for authorizing the key and another one for unlocking the door. Much like LACs, the LCs communicate with other LCs, illustrated in Figure 1. On this level the communication is described much more in detail. The design view is stored in a custom made proprietary tool. No automatic transformation is done between the architecture model containing the logical view and the tool containing the design view. Both are complete views of the whole system architecture.

From the tool containing the design view a number of things are generated. Firstly, for in-house software development, model shells are generated to be filled with behavior by the in-house software developers. From these models, code is generated. Secondly, for functionality or parts of functionality to be implemented by suppliers, requirement documents and other specifications are generated. Artifacts used for ECU-integration and network communication are also generated from the tool. Because of the number of artifacts generated from this tool, the model within needs to be kept up to date as otherwise integrating and building the software would fail.

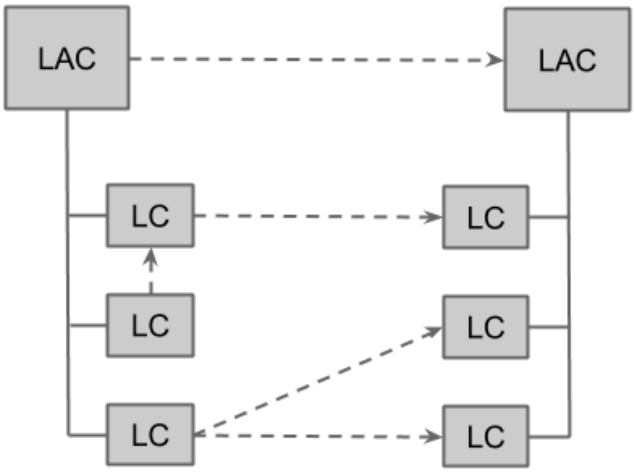

Fig. 1. *Image depicting two LACs, the LCs assigned to them, and the communication between them. Note that LCs can communicate both within their LAC as well as with LCs in other LACs.*

F. Deployment and Efficiency

The LACs are deployed to different physical ECUs, each ECU belonging to a domain. Domains are used in the car to group ECUs by their main purpose. For example, the propulsion domain containing nodes for engine control and gearbox among others. Each domain has its sub-network in the car to make communication between nodes in the same domain cheaper and faster. These domain specific networks are connected via gateway nodes, called domain masters, to a backbone. This subdivision affects the efficiency of transferring data between the nodes. For instance, inter-domain communication is slower and less efficient than intra-domain communication, resulting in a higher cost as seen in Fig. 2.

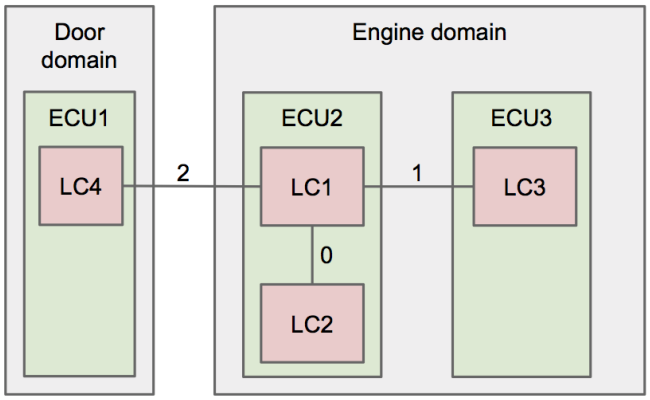

Fig. 2. *Figure illustrating that different types of communication have different efficiency cost. Intra-ECU communication is the cheapest, inter-ECU but intra-domain more expensive and inter-domain communication the most expensive. Actual numbers are for illustrative purposes, the real ratio between them might be different.*

If an LAC is deployed to an ECU architectural rules says that its LCs should be realized on the same ECU, as seen in Fig. 3. Having LCs that need to communicate with each other on different ECUs means that communication between them needs to pass via the networks within the car. Therefore, the architectural decisions of where to place LACs and LCs have an impact on the communication efficiency within the car. Because of backward compability, refactoring things that affects signaling on the internal network in the car, between ECUs, is diffifult and avoided as long as possible.

III. Methodology

To answer the research questions presented in Section I, we conducted an exploratory case-study [12] at the electrical development department at VCG. The study was divided into the following four steps.

Firstly, we identified a relevant and typical kind of ATD (according to the stakeholders) in the VCG context. Secondly, we investigated what was the impact (interest) that such ATD items had on the software development process and the system quality. This activity provided results to answer RQ1 and RQ2.

For these two steps, we conducted three semi-structured interviews as well as two workshops with three domain experts working as architects at VCG. Based on the results of these interviews, we identified a particularly relevant ATD item in the VCG context (to answer RQ1) and their interest (RQ2).

In the third step we developed a visualization tool, based on a literature review of the state of the art in software visualization (e.g. by surveying SLRs and recent work on the topic such as [2], [6]), to provide a suitable representation of the ATD items and their interest previously elicited. The

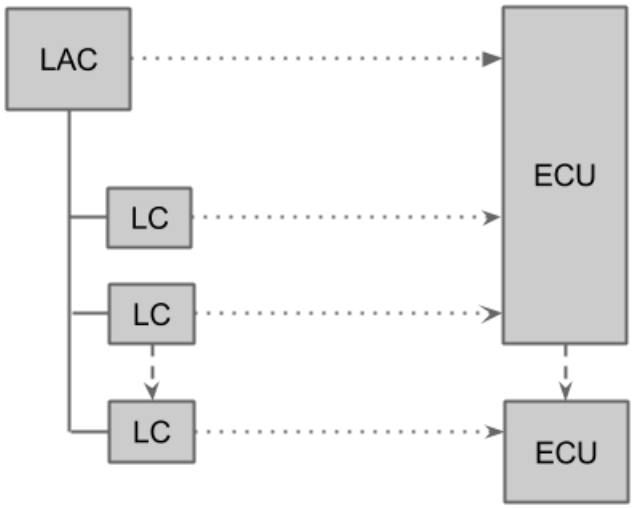

Fig. 3. *Image depicting an LAC and its assigned LCs being deployed to two different ECUs. The dotted lines show how the components are deployed and the dashed lines the communication between components.*

tool automatically collect the necessary data from models and databases, and compare the high-level architecture with the detailed design, calculating the technical debt and visualize it. An early version of the tool was also shown and discussed in a workshop at VCG with two domain experts, in order to get early feedback on the current direction regarding the kind of visualization and the attributes to be visualized. Such activity was aimed at answering RQ3.

Finally, we evaluated the ATD items, their interest and the visualization tool in three separate interviews with three domain experts at VCG (in order to increase the reliability of our results through triangulation, as recommended in [12]). During the validation interviews, the tool was demoed for domain experts, followed by a semi-structured interview with the purpose to asses the usefulness of the metrics chosen and understandability of the visualization presented by the tool.

In particular, in the validation interviews we asked questions with respect to the following aspects:

- Participants' background, e.g. "What is your role with respect to the architecture"
- Feedback on the debt items, e.g. "Does the visualized debt item reflect a situation that occurs in the system?"
- We described the metrics used for defining and representing the interest and we asked questions such as "Do you think that it would be useful to identify the interest caused by additional dependencies due to the LCs having dependencies that does not exist in their LAC?"
- In the visualization part, we asked questions such as "Is the interest difference between debt-items clearly communicated?"

IV. Results

In this section, we report our results for each of our three research questions.

A. RQ1: What is a Typical ATD Item for the Automotive Industry?

From our initial interviews we found that the stakeholders considered valuable to identify a specific ADT item originating from architecture violations: we called it *Misplaced LC*.

The ATD *Misplaced LC* comes from the fact that although the architectural rules say that all the LCs contained in one LAC should be realized on the ECU where the LAC is deployed, the architecture group have no means to technically enforce this. This can lead to LCs being deployed on different ECUs than the ones that were intended by the architects, resulting in non-allowed dependencies between different domains (the actual technical debt). An illustration of this can be seen in Figure 4.

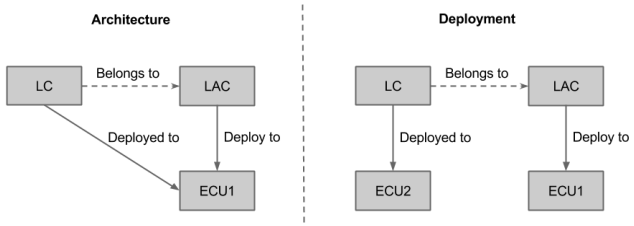

Fig. 4. *Illustration of the Debt Item (Misplaced LC) depicting an LC, assigned to an LAC, deployed to a different ECU than its LAC states. The left part shows the intended deployment and the right part the actual one.*

As the cost of communication differs depending on if it is internal on the ECU, if it is between ECUs or if it's across the domains, such a violation may result in a change in the amount of communication over the network have an impact on efficiency, as seen in Fig. 5.

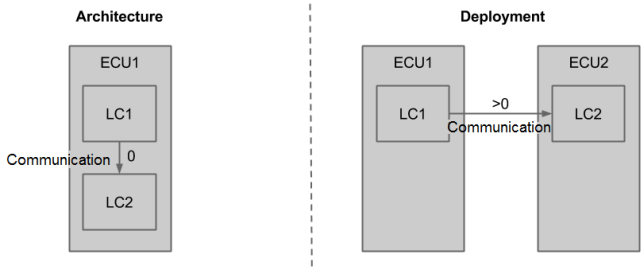

Fig. 5. *Illustration of the effect of the Debt Item (Misplaced LC) depicting LC2 introducing a dependency from ECU1 to ECU2 by being misplaced. The left side shows the intended architecture while the right side displays the actual architecture. Note the discrepancy in communication cost.*

B. RQ2: What is the Impact, or Interest, of such ATD Item?

For *Misplaced LC*, an LC is deployed to an ECU that it should not be deployed to according to the architecture models. This may change the cost of communication since communication between ECUs or between domains is more costly than communication within an ECU, as shown in 2.

To get the interest for this debt we look at the dependency cost for an LC if it was deployed in accordance with the architecture and then subtract it from the actual dependency

cost, as seen in 1. The interest shows whether the architectural violation increases or decreases the dependency cost for an LC.

$$\mathbf{LC}_{cost}(lc) - \mathbf{LC}_{cost}(lc') \tag{1}$$

To perform the calculation, shown in 1, a number of different steps need to be conducted. The initial step is to identify the LCs violating the architecture by being deployed to a different ECU. The next step is that for each identified LC look at its current location and then look at its location as if it were deployed in accordance with the system architecture. This provides us with two different LCs for each LC we initially identified, as seen in 2.

$$\begin{aligned} &\mathbf{lc} : \text{LC on its current location in the system} \\ &\mathbf{lc}' : \text{LC if it was located in accordance with the architecture} \end{aligned} \tag{2}$$

After this, the next step, shown in 3, is to get a set of all dependencies for the LC. This set of dependencies will be equivalent for *lc* and *lc'* since even if they are placed in different ECUs, their dependencies have not changed.

$$\mathbf{LC}_{dependencies}(lc) : \text{All dependencies for a given LC} \tag{3}$$

What actually differentiates *lc* and *lc'* is whether the dependencies between their ECUs are intra-ECU, inter-ECU or inter-domain, and depending on what category they fall under each dependency is assigned a corresponding weight as seen in 4. Note that the values for weighting are not a part of the proposed metric since they likely differ between different systems; zero to two are used for illustrative purposes and further discussion about the weighting values can be found in the validation and discussion sections.

$$\begin{aligned} &\mathbf{W}(d) : \text{The weight for a given dependency} \\ &\mathbf{W}(d) = \begin{cases} 0, \text{if d communication is intra-ECU} \\ 1, \text{if d communication is inter-ECU} \\ 2, \text{if d communication is inter-domain} \end{cases} \end{aligned} \tag{4}$$

To get the cost for an LC, all the dependencies for that LC are iterated over, see 3, and for each dependency the weight is calculated, see 4, and summed to get the dependency cost for that LC, as seen in 5.

$$\mathbf{LC}_{cost}(lc) = \sum_{d}^{\mathbf{LC}_{dependencies}(lc)} \mathbf{W}(d) \tag{5}$$

Equation 5 is then performed on both lc and lc' to get the dependency cost for both the intended architecture as well as the actual architecture; the difference between both make up the efficiency interest for a misplaced LC. The final mathematical formula, seen in 6, corresponds to the initial interest in Equation 1.

$$\left(\sum_{d}^{\mathbf{LC}_{dependencies}(lc)} \mathbf{W}(d)\right) - \left(\sum_{d}^{\mathbf{LC}_{dependencies}(lc')} \mathbf{W}(d)\right) \tag{6}$$

C. RQ3: How can the ATD and Its Interest be Visualized Together to Aid the Stakeholders' Awareness?

To give the stakeholders a comprehensible way of understanding the efficiency interest described above a visualization technique was developed, seen in Fig. 6.

The first step in creating the visualization was to decide what part of the system that had to be visualized. Since both of the proposed metrics, detailed in Section IV-A, measure the interest of individual architectural components, the conclusion was that the visualization needs to map components to their measured interest.

2D boxes are used to represent a single components in the architecture. Since metrics are per component, the relations between each component and the structure of the system are not necessary to show the efficiency interest. This is in contrast to other similar visualization techniques using boxes, such as treemaps [5], where boxes are placed inside boxes to visualize a hierarchy.

In order to present quantified data for the components, *i.e.* the visualized metric, the visualizer is influenced by space filling techniques [17]; the side length of the boxes are proportionate to the measured efficiency interest, making it possible to visually compare the interest of different components, and the colour reflects if the interest is positive or negative. In further accordance with space filling techniques, the boxes are scaled to fit the full area of the display as well as possible.

Finally, the described visualization technique provides an overview of the system, however, to get a more detailed view of the system the boxes were made clickable; by clicking the boxes, an additional window provides the stakeholder with additional, more detailed, information about the component that the box represents.

V. Discussion

A. Implications for academia

The study reports a novel solution in the field of Technical Debt management, by showing how ATD items and their interest can be combined and visualized (answering RQ3). We also show how such visualization is useful for the involved stakeholder at VCG, to be aware of the presence of ATD and the impact of such sub-optimal solutions on important qualities. We also contribute to the state of the art of TD by proposing a visualization technique that can be used for showing the amount of TD and its interest currently paid by the organization.

We found that the awareness of the existing dependencies is particularly of interest for the architects in this context (which answers RQ1), especially if combined with other information about the impact of such dependencies on the efficiency of the system (which would answer RQ2). We therefore see an opportunity for future research to further investigate how different impacts can be shown together with respect to the same ATD item, to provide information on the full picture about ATD items and therefore aiding the decision making on practically hard problems such as incurring or removing ATD.

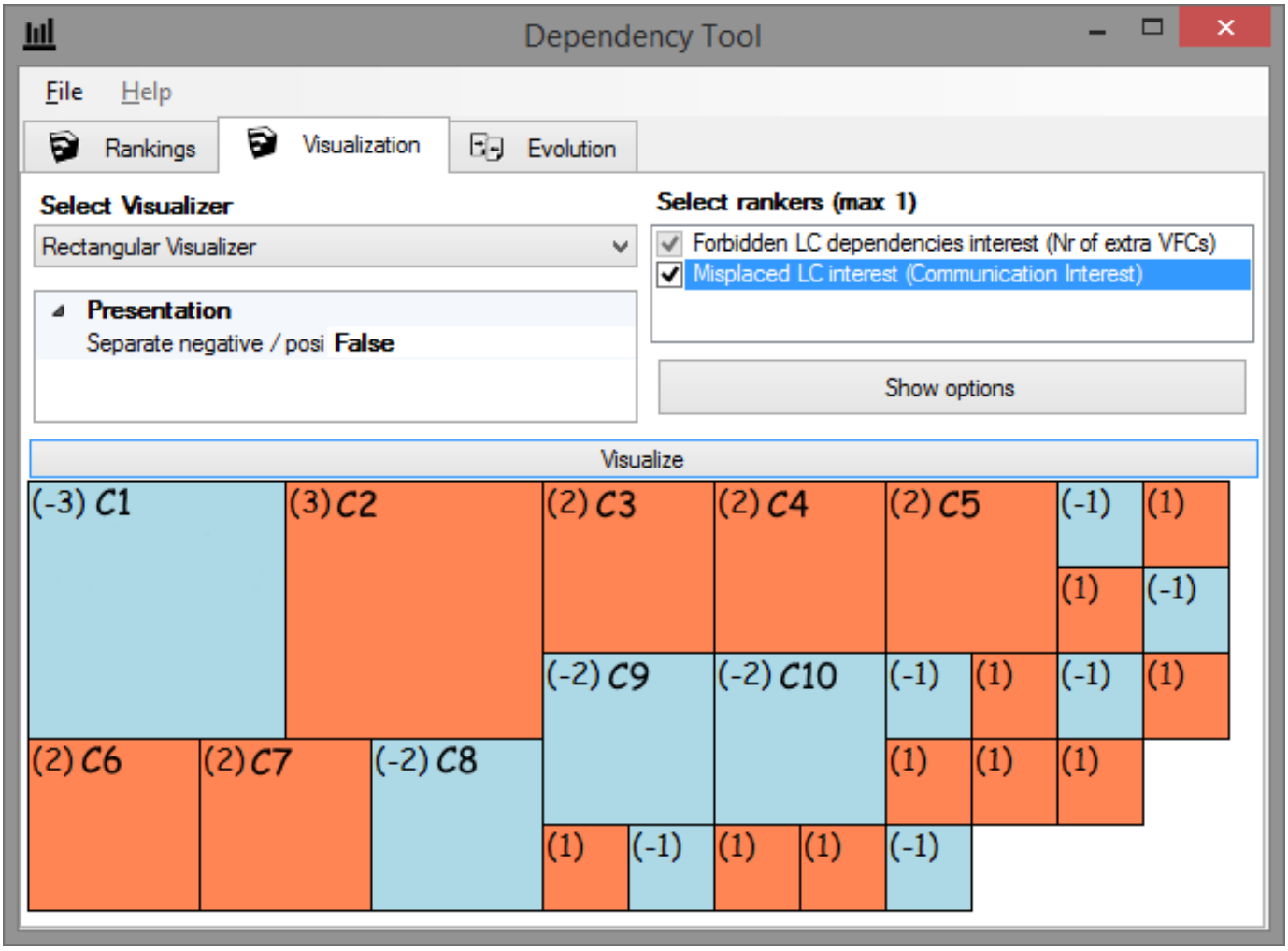

Fig. 6. *Screenshot of the developed tool, showing the visualization. It shows components affecting the system negatively, red boxes, and components affecting the system positively, blue boxes. The size of each box is proportionate to their efficiency interest and the text within indicates their measured interest, within parentheses, and their unique identifier, in this case C1,C2,...,C5.*

Our approach analyzed architecture models on different levels to find the ATD items. This contributes to the existing body of knowledge, since most of the existing tools use source code as input [7], and we show how ATD can exist not only between implementation and architecture, but also between different levels of abstraction of the same architecture.

B. Implications for industry

There are a couple of factors that can explain this discrepancy between the high-level architecture and the design of the implemented system. The different levels of the architecture and design are done in different groups, the high-level architecture in the system architecture group and the detailed design work is spread out among the various development groups responsible for their sub-systems, nodes and components. There is no transformation between the high-level architecture and the lower levels leaving room for having differences between the high-level and the design level.

However, a stronger connection between the two is not desirable in all situations, as we have found in previous work [4]. For example, it would put a stricter requirement on the formality of the high-level architecture model that the architects feel would impede their creative work and possibilities to explore future solutions [4]. It is possible that some of the differences between the high-level architecture and the detailed design are because of conscious decisions, such as because IO-pins for controlling a certain actuator is physically on an ECU forcing an LC to be realized at that particular ECU. Something that the system architecture group might not be aware of as it is impossible for them to have such detailed knowledge of all parts of the system. In other cases it is possible that inconsistencies exist by accident.

Efficient communication in the car is important for three reasons. Firstly, a lot of functions within the car have strict requirements on timing and latency, requirements that get harder and more costly to meet as the amount of communication over the networks increases. Secondly, for all automotive OEMs

that mass produce vehicles, the component cost is important. A slight increase in the cost for a single component quickly accumulates to a big sum when hundreds of thousands of vehicles are produced. It is, therefore, desirable to keep the communication on the networks down to avoid running out of bandwidth and being forced to upgrade the network. Finally, communication that is once scheduled to go on the network is hard and sometimes impossible to change because strict requirements on backward compatibility. It should be possible to upgrade individual nodes without having to worry about breaking functionality.

Therefore, a tool such as the one developed during this research project, is valuable for industry to be able to identify quickly and prioritize where to put time and resources to review and, if deemed needed, fix the inconsistencies. Having a visual tool in which the stakeholders quickly can get an overview was shown to be valuable.

C. Limitations and Threats to Validity

To address potential limitations in the study, and the trustworthiness of its results, the validity threats detailed in the guidelines by Runeson and Host [12] are used. The threats identified for this thesis are divided into three categories: Construct validity, External validity, and Reliability; no internal validity threats were identified since this thesis does not aim at providing an empirical proof of a cause/effect relationship, but rather a way to represent it (assuming its existence, as declared by the stakeholders). For each category, this section presents the identified validity threats.

D. Construct Validity

The part of this study that relies on construct validity is the evaluation of the proposed methodologies. Since the evaluation is performed as a series of semi-structured interviews, it is important that the interviews address the quality of the result without bias from the researchers. Therefore, to minimize potential threats to construct validity, the interviews are conducted anonymously, encouraging discussion, and specifically enables the interviewees to criticize the proposed methodologies.

E. External Validity

Since there is only one studied system, the resulting methodology is entirely based on such studied case and could therefore potentially be limited to similarly designed architectures, e.g. where an intended architecture is realized as logical components deployed to ECUs. Similarly, the interviews were held with four people from the company, all knowledgeable in the studied system. This likely produces a homogeneous result and could mean that there are other aspects important to the system that the interviewees did not consider. In order to compensate, follow up meetings were used to clarify and further discuss interesting concepts, leading to new insights. Regarding the efficiency interest, the impact of dependencies may differ in other systems. However, the weighting function can be calibrated in order to address other systems' specific requirements. The weights used in the proposed method had an illustrative purpose, and can easily be adjusted for the intended architecture. As for efficiency, other contexts, especially related to embedded systems, may consider similarly important such quality. In summary, although the actual weights used in this case should be re-calibrated, we see how a similar visualization approach can be used in other contexts in discovering ATD items based on non-allowed dependencies and visualizing them together with the efficiency interest.

F. Reliability

The initial interviews are semi-structured and entailed exploratory questions on the architectural structure and possible metrics. Although the interviewees are domain experts, it is not entirely unlikely that future interviews at the same company would reveal additional or alternative strategies for addressing dependencies in the system. Furthermore, due to the complexity of the studied system and the understanding of it being derived from interview interpretation, discrepancies between reality and theory may be introduced. To account for both of these validity threats, continuous interviews are made to further explore ideas and confirm previously established notions, extracting as many strategies as possible and correcting eventual misunderstandings. The first author of this paper, also involved in many activities during the research process, is also working at VCG, which greatly minimizes this threat.

G. Related Work

The phenomenon of ATD has only recently received the attention of the research community [7], [16], [8]. It was difficult, therefore, to find extensive previous research tackling the problem of prioritizing ATD items based on their effects in real contexts, which motivated our exploratory study.

Few single case-studies were related to code-debt or code-smell [18], [14]. The work done by Sjøberg et al. shows that some code-smells don't create extra maintenance effort: the implications are that not all the smells or the architecture quality problems identified in literature have necessarily a big impact on maintenance effort, but might have other kinds of impacts.

The related work that can be considered as the closest to our study is the case-study conducted by Nord et al. [9], where the authors studied a specific case in depth, modeling and developing a metric based on architectural rework that would help deciding between different paths leading to different outcomes. However, such study does not take in consideration interest related to quality attributes such as efficiency and it does not address the problem of visualizing dependency violations with respect to their interest.

Another work existing in literature covers the impact of changes to the complexity and coupling properties of automotive systems [3]. Such paper is relevant as it covers similar problems in the automotive domain. However, rather than investigating the impact of dependency violations, it focuses on the impact of architectural changes on the increment of quality metrics such as coupling and complexity. Even though

such metrics are related to the presence of dependencies, they don't focus on the specific ones that are not supposed to be implemented in the system (according to the architectural rules), but rather on an overall number of dependencies (that might be sub-optimal). Also, although the study takes in consideration weights for the communications between ECUs, it does not address the efficiency interest and the visualization of the impact due to the non-allowed dependencies.

VI. Conclusion and Future Work

In our research project we have started to investigate what ATD is relevant for the automotive domain. In this paper we have studied one kind of ATD: the deployment of software components violating the high-level architecture specification of the allowed dependencies. We have also investigated the impact (interest) of such sub-optimal solutions, which results in decreased efficiency of the communication within the vehicle. Such system quality is important for cars (and other embedded systems), since efficient communication and utilization of the networks within the car is crucial for the functionality of a car, but also for the long-term profitability as each increase in component cost is costly for a mass produced product.

In this paper, we present a novel approach for visualizing the debt and the interest together. Such visualization helps the stakeholders identifying and prioritizing ATD, by understanding the impact of different ATD items (specifically non-allowed dependencies) on efficiency. The validation interviews with the stakeholders at VCG confirmed that such tool would be valuable for architects and other stakeholders.

Our approach analyzed architecture models on different levels to find the ATD items. This contributes to the existing body of knowledge, since most of the existing tools use source code as input, and we show how ATD can exist not only between implementation and architecture, but also between different levels of abstraction of the same architecture.

In the future we plan to continue our research on what other ATD items can be found and what other impact measures can be combined together for having the complete picture of ATD and its interest, which would greatly help the decision making of the main stakeholders. We also plan to investigate the reasoning behind the architectural violations, whatever they are intentional or not, and whatever or which the architectural violations benefit the final product. Finally, stakeholders that were part of the early evaluation of the tool said that it would be valuable to see the evolution of the debt. They wanted to see if the tendency is that the debt increases or decreases over time, as well as see what impact planned or potential future changes would have. As seen on the title of the last tab in Fig. 6, the work of adding such functionality have started but, at the time of writing this paper, the functionality has not yet reached such status that it could be evaluated.

Acknowledgements

We thank Volvo Car Group for their involvement in the research and acknowledges support by the ASSUME-project Vinnova and the Software Center initiative (software-center.se).

References

[1] N. Brown, Y. Cai, Y. Guo, R. Kazman, M. Kim, P. Kruchten, E. Lim, A. MacCormack, R. Nord, I. Ozkaya, R. Sangwan, C. Seaman, K. Sullivan, and N. Zazworka. Managing technical debt in software-reliant systems. In *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research*, FoSER '10, pages 47–52, New York, NY, USA, 2010. ACM.

[2] P. Caserta and O. Zendra. Visualization of the static aspects of software: A survey. *Visualization and Computer Graphics, IEEE Transactions on*, 17(7):913–933, July 2011.

[3] D. Durisic, M. Nilsson, M. Staron, and J. Hansson. Measuring the impact of changes to the complexity and coupling properties of automotive software systems. 86(5):1275–1293.

[4] U. Eliasson, R. Heldal, P. Pelliccione, and J. Lantz. Architecting in the Automotive Domain: Descriptive vs Prescriptive Architecture. In *12th Working IEEE / IFIP Conference on Software Architecture*, Montreal, Canada, May 2015.

[5] B. Johnson and B. Shneiderman. Tree-maps: A space-filling approach to the visualization of hierarchical information structures. In *Proceedings of the 2Nd Conference on Visualization '91*, VIS '91, pages 284–291, Los Alamitos, CA, USA, 1991. IEEE Computer Society Press.

[6] T. Khan, H. Barthel, A. Ebert, and P. Liggesmeyer. Visualization and Evolution of Software Architectures. In C. Garth, A. Middel, and H. Hagen, editors, *Visualization of Large and Unstructured Data Sets: Applications in Geospatial Planning, Modeling and Engineering - Proceedings of IRTG 1131 Workshop 2011*, volume 27 of *OpenAccess Series in Informatics (OASIcs)*, pages 25–42, Dagstuhl, Germany, 2012. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[7] Z. Li, P. Avgeriou, and P. Liang. A systematic mapping study on technical debt and its management. *Journal of Systems and Software*, 101(0):193 – 220, 2015.

[8] A. Martini, J. Bosch, and M. Chaudron. Architecture technical debt: Understanding causes and a qualitative model. In *Software Engineering and Advanced Applications (SEAA), 2014 40th EUROMICRO Conference on*, pages 85–92, Aug 2014.

[9] R. Nord, I. Ozkaya, P. Kruchten, and M. Gonzalez-Rojas. In search of a metric for managing architectural technical debt. In *2012 Joint Working IEEE/IFIP Conference on Software Architecture (WICSA) and European Conference on Software Architecture (ECSA)*, pages 91–100.

[10] M. Peter and E. de Quincey. A gentle overview of software visualization. *PPIG News Letter*, pages 1 – 10, 2006.

[11] B. A. Price, R. M. Baecker, and I. S. Small. A principled taxonomy of software visualization. *Journal of Visual Languages & Computing*, 4(3):211 – 266, 1993.

[12] P. Runeson and M. Hst. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14(2):131–164, 2009.

[13] M. Shahin, P. Liang, and M. A. Babar. A systematic review of software architecture visualization techniques. *Journal of Systems and Software*, 94(0):161 – 185, 2014.

[14] D. I. Sjoberg, A. Yamashita, B. C. Anda, A. Mockus, and T. Dyba. Quantifying the effect of code smells on maintenance effort. 39(8):1144–1156.

[15] D. Tamburri, P. Kruchten, P. Lago, and H. Van Vliet. What is social debt in software engineering? In *Cooperative and Human Aspects of Software Engineering (CHASE), 2013 6th International Workshop on*, pages 93–96, May 2013.

[16] E. Tom, A. Aurum, and R. Vidgen. An exploration of technical debt. 86(6):1498–1516.

[17] T. von Landesberger, A. Kuijper, T. Schreck, J. Kohlhammer, J. van Wijk, J.-D. Fekete, and D. Fellner. Visual analysis of large graphs: State-of-the-art and future research challenges. *Computer Graphics Forum*, 30(6):1719–1749, 2011.

[18] N. Zazworka, A. Vetro, C. Izurieta, S. Wong, Y. Cai, C. Seaman, and F. Shull. Comparing four approaches for technical debt identification. 22(3):403–426.