CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Visualization of Software Architecture based on stakeholders' requirements

Empirical investigation based on 4 industrial cases

Bachelor of Science Thesis in Software Engineering and Management

ANNA GRADULEVA
MARJAN ADIBI DAHAJ

Department of Computer Science and Engineering
UNIVERSITY OF GOTHENBURG
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2017

CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Visualization of Software Architecture based on stakeholders' requirements
An empirical investigation of stakeholders' requirements towards Software Architecture Visualization based on 4 industrial cases.

Anna Graduleva
Marjan Adibi Dahaj

Supervisor: Truong Ho-Quang
Supervisor: Michel Chaudron
Examiner: Jan-Philipp Steghöfer

University of Gothenburg
Chalmers University of Technology
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
UNIVERSITY OF GOTHENBURG
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2017

Visualization of Software Architecture based on stakeholders' requirements

Multiple case study

Anna Graduleva
Department of Computer Science and Engineering
University of Gothenburg
gusgraduan@student.gu.se

Marjan Adibi Dahaj
Department of Computer Science and Engineering
University of Gothenburg
gusadibma@student.gu.se

***Abstract* -- Considering the rapid growth of software systems and consequential difficulties with development, evaluation, maintenance and reengineering, there is an emerging demand for effective means for communication of software architecture. One of such techniques is Software Architecture Visualization (SAV). However, visualization of an entire architecture is overwhelming to the user and thus possesses little value. Therefore, it is essential to determine possible stakeholders and identify what visualization is preferred by each. However, present research lacks support from industry practitioners in determining the relationship between stakeholders and levels/types or visualization. In this study qualitative data gathered from interviews with Volvo, Ericsson and Tetra Pak is analyzed to determine information need, preferred techniques, tools and levels of abstraction depending on a stakeholder. Requirements of the stakeholders were compared and contrasted to each other, as well as literature results. Lastly, this paper presents complementary or substitutionary visualization techniques based on a stakeholder and lists practical implications that could be useful to SAV practitioners and tool vendors.**

Keywords – software architecture, software architecture visualization, stakeholders.

I. INTRODUCTION

With a rapid growth of complexity of software system it becomes more difficult to undertake development-related activities that require a degree of system comprehension [1]. Consequently, this spiked an interest in techniques and tools that would aid understanding and communication of system's structure, behavior, and evolution of the software [2]. Software Visualization (SV) attracted attention of researchers and practitioners, due to the fact, that visual representation supports more effective comprehension of large amount of data than text-based descriptions [3]. Software Architecture Visualization (SAV) in particular became central within SV research since architecting process is prominent throughout system's lifecycle [4], including activities such as "analyzing, synthesizing, evaluating, implementing, and evolving architecture"[5] .

SAV is a well-established research field that has been growing for the past two decades [2], with primary focus on benefits of SAV, SAV techniques and supporting tools. Considering the variation of interest in and purpose of employing SAV, the research produced a vast number of different techniques, ranging from industry standard, Unified Modelling Language (UML), to innovative 3D metaphor-based visualizations. Many of new techniques are proposed with complementing visualization tools. Understandably, manual visualization of Software Architecture (SA) may not be of interest to practitioners due to the size and complexity of today's systems, and are generally substituted by automatic and semi-automatic tools. The benefits of using SAV tool are considerable, as they provide "significant value in understanding large software architectures and supporting architectural maintenance and evolution, quality assessment, communication with stakeholders, and strategic product planning"[5] as well as reduced costs associated with development and evolution of software [7].

Existing research [1,2,5,6] overviews and evaluate a number of tools and techniques that support different activities but there is still an insufficient number of empirical research in close co-operation with practitioners that would demonstrate SAV application in the industry.

Besides application of SAV in the industry, the SV field lacks research on the difference of techniques, tools, and abstraction level of visualization depending on stakeholders involved in software development process interests. Visualization of software architecture alone does not provide a highly useful overview of software architecture, since due to system's complexity, a single view covering all aspects of the system can become quickly overwhelming. What is more, different stakeholders, whose concerns are separated, rarely require same visualization [5, 8]. Current research acknowledges the difference in needs of stakeholders when it comes to SAV, but it does not specify or advice on specific methods, types of visualization, or levels of detail. For example, Telea et al. [5] recognizes that non-technical stakeholders can be more concerned with evolution of system over time than low-level developers and require abstracted visualization, and then assess to what extent current tools support these general needs.

The research does give a general understanding of difference between different stakeholders' requirements for SAV, but for the most part, it is not demonstrated by examining its application in the industry, that was also pointed out by a number of studies [2, 6].

Therefore, visualization of software architecture without targeting a specific stakeholder group provides reduced benefit and poses a risk of negative effects associated with low system comprehension. Carpendale and Ghanam [8] stress the importance of defining stakeholders when it comes to SAV: "defining the audience of the architecture visualization plays a pivotal role in determining what to visualize and how to visualize it".

The structure of this paper is as follows: Section I introduces the general concepts of SAV and defines a problem that is to be addressed; Section II specifies purpose of the study and lists research questions; Section III describes case companies; Section IV discusses the method; Section V is a literature review, and Section VI displays gathered interview results; Section VII includes discussion of results; and conclusion in Section VIII summarizes paper's findings.

II. Purpose of the Study

Considering increasing interest in SAV of both researchers and practitioners, and lack of empirical investigations of SAV application within industry, the purpose of this study is:

1. To determine what is the state of SAV employment by practitioners based on stakeholder type, including demand for SAV, difference in techniques, tools, and, most importantly, difference of required level of abstraction;

2. To provide practical implications of scientific findings that could assist practitioners in adoption of SAV based on stakeholder type, including appropriate techniques and, most importantly, appropriate level of abstraction.

The results of this thesis is firstly: filling the gap in current knowledge by investigating current SAV practices based on 4 studied cases, with focus of different stakeholders' requirements for level of abstraction, tool support, and appropriate techniques; and secondly: provide practical implications for practitioners that seek to adopt SAV within their projects, containing recommendation to which techniques, tools, and, most importantly, level of abstraction are demanded from different stakeholders. Both contributions will be based on studying 4 industrial cases in conjunction with existing literature on the subject. The industrial cases include two separate series of interviews with Ericsson, a series of interviews with Volvo Cars, and a series of interviews with Tetra Pak.

Six research questions were defined that this paper aims to answer:

RQ1. What is the current demand for SAV in the industry depending on a stakeholder?

RQ2. What is the information need of different stakeholders towards SAV?

RQ3. What techniques of SAV can be employed depending on a stakeholder?

RQ4. What is the level of abstraction required from SAV depending on a stakeholder?

RQ5. What type of tools are used for SAV depending on a stakeholder? (automatic, semi-automatic, manual)

RQ6. What are the reasons for not employing SAV in the industry?

III. Case Companies

A. Volvo (Case 1)

System designer, software developer, and a test engineer were interviewed to mainly determine their information needs when it comes to architectural description, which in this case, was stored in "the database". The study, these interviews were part of, concentrated on information need and requirements towards software architecture visualization, while omitting information concerning current employment of SAV, structure of teams and interviewees' experience to a large extent. It was briefly mentioned, that software developer worked as a part of development team, consisting of 8 developers, and had at least 4 years of development experience while working with "the database". System designer did not provide information about whether he works as a part of a team, and its composition, but he had over 2 years of working experience with their architecture description tool. Lastly, test engineer had at least 3 years of experience of working with "the database", but offered no information about his/her assignment to any teams.

B. Ericsson (Case 2)

Three design architects, a system architect and a designer were interviewed for case 2 study, which concentrated on information need of architects, with particular focus on what constitutes a software entities vital to visualize.

All of the interviewed stakeholders had over 10 years of experience and were part of different teams, which ranged from 6 to 10 people. Their experience with UML, on the other hand, varied greatly, ranging from less than a year to over ten years of experience. Lastly, it is important to note, that 2 interviewed architects also work as developers that can influence their information need or level of abstraction required.

C. Tetra Pak (Case 3)

Case 3 study contained interviews with 8 stakeholders: system and software architects, design architect, two developers, team and project managers, and a test engineer.

All the stakeholders, except for system architect and managers are distributed between 2 teams, which consist out of 6 people each. Majority of the stakeholders have over 10 years of experience, except for test engineer, who has 2 years of experience. Lastly, although 3 of stakeholders have responsibilities that deal with architectural design, majority of their time is occupied with development that can be reflected in the data.

D. Ericsson (Case 4)

As part of this case, 5 stakeholders were interviewed, including system and design architects, a manager, a developer, and a function tester. All of these stakeholder work as part of separate teams, except for system and design architect, who work in a same team consisting of 3 architects. The developer works in a cross-functional team, consisting of 7 people, the managers oversees several teams at the same time, and function tester is not assigned to any particular team. System architect and software developer have approximately 20 years of experience, while design architect and the manager have 9 years of experience, and the tester has 5 years of experience.

E. Additional Comments

Both cases of Volvo (case 1) and Ericsson (case 2), are special cases, since case 1 concerns visualization of electrical architectures in the automotive domain, while case 2 was limited to interests of mainly system and design architects, with no participating developers or managers. Additionally, case 1 participants described their information needs and possible improvements to visualizations, but did not cover what were their current SAV practices, such as currently used techniques and tools.

IV. METHODOLOGY

In this section, the process of defining research questions, conducting literature review and interviews, data condensation and data analysis will be described.

A. Why Case study?

A number of existing research [2, 6] recognized the need of examining SAV in industrial setting, proposing controlled experiment or case study methods. However, it can be rather difficult to assemble a group of highly motivated experiment participants from the industry [9], as well as high resource and effort cost [10], which are currently cannot be met.

Generally, a case study investigates "contemporary real-life phenomenon through detailed contextual analysis of a limited number of events or conditions, and their relationships" [14]. This can be a more light-weight process, compared to controlled experiments, as it requires a smaller number of participants. Case study also proved to be advantageous, when a "holistic, in-depth investigation is required" [14].

Multiple Case study will allow us to critically analyze SAV application in the industry in respect to the different contexts presented in "Case Companies" section. Gathering and analyzing data from multiple cases decreases bias and ensures internal validity of the research [11]. Empirical qualitative data can also give an opportunity to form new relationships between pieces of the data, for example, SAV application in a context of a specific company and maturity of development practices of the company. In this case, qualitative data has an obvious advantage due to the need of obtaining rich information of the context in which SAV takes place. This context can be related to social and human behaviors and might require a flexible method of data gathering, such as an interview.

This case study possesses characteristics of explorative study as it attempts to investigate what is the current state of SAV in the industry and determine what kind of visualization is required based on a stakeholder type. However, it also attempts to analyze the differences of requirements between various stakeholders, as well as difference of requirements of

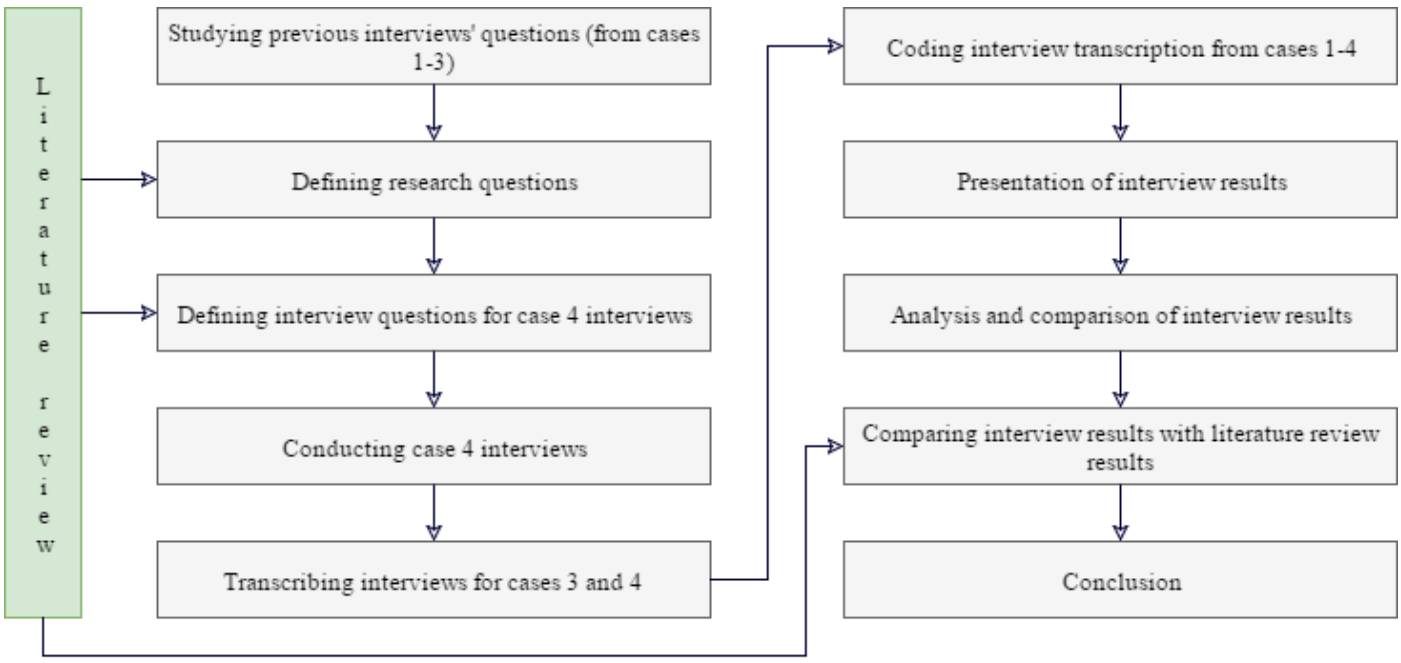

Figure 1. Overview of the process of defining research questions, gathering data and data analysis.

same stakeholders across different companies.

B. Process Outline

It is important to note, that the data set that was analyzed to answer research questions consisted of 4 separate data sets, 3 of which are: 3 interviews conducted with Volvo Cars (case 1), by Florence Mayo and Nattapon Thathong [46], 5 done with Ericsson (case 2) by Filip Brynfors [47], 8 interviews done with Tetra Pak (case 3) by Truong Ho Quang in June 2016. The last data set comprised 5 interviews carried out by authors of this paper with Ericsson employees in April 2017.

The process of problem elicitation, definition of research questions, conducting literature review, and gathering of qualitative data was divided into 5 steps. An overview of the process is also displayed in figure 1.
Step 1: Problem elicitation by reviewing related literature. Definition of research questions based on interview questions from pre-existing dataset without knowledge of interview results to avoid bias. Step 2: Conducting a literature review of related research, which would later be compared with interview results. Step 3: Composing a list of interview questions based on research questions and conducting interviews with participants of case 4. Step 4: Transcribing and coding of the interviews. Step 5: comparing and contrasting the results of interviews with similar and conflicting literature.

C. Literature Review

Preliminary literature review was carried out with aims of identifying research gap, formulating relevant research questions and motivating the research. Once research questions were identified, a more extensive literature review was conducted, the results of which later on would be compared with qualitative results of interviews.
Manual search of academic papers and sorting was performed, resulting in 37 papers, mostly published between 2003 and 2016, with some earlier publications in 1990 and 2000.

Since literature review included 4 different subsections (stakeholders, benefits, techniques, and tools) which were based on reviewing different types of research, inclusion criteria was broad. For "Tools" and "Techniques" subsections, for instance, it was important that a presented tool or technique was sufficiently evaluated. For "Techniques" section it was particularly important to present contrasting views on same techniques and approaches in order to display advantages and disadvantages of their application. Overall, most of the studies were published in last 15 years, with some exceptions for taxonomies, which were published earlier on.

D. Data Collection

After a gap in research was identified, a set of research questions were defined based on literature review and interview questions of previous interviews. However, it was important to avoid bias, and therefore, the research questions were formulated without reading the interview transcripts. Instead, the interview questions were carefully studied, after which the research questions were defined. As a result, definition of research questions was independent from gathered data, which decreased likelihood of validity threats emerging.

The final data set consists of 4 separate data sets, which will be analyzed together: 3 interviews done with Volvo (case 1), 5 interviews done with Ericsson (case 2), 8 interviews done with Tetra Pak (case 3), all of which were carried out in the year of 2016 by different researchers, other than authors of this paper. The fourth dataset (case 4) consists of 5 interviews done with Ericsson by the authors of this paper in 2017, April. These companies were chosen because of difference in terms of domains, team size and development practices. This gives an opportunity to analyze the data in two layers: how SAV application differs from one stakeholder to another in the same context; and how SAV application differs for the same stakeholder type in the different contexts. These companies' domains, sizes and organizations may lead to vastly different employment of SAV that would allow researchers to account for different perspectives and make the results of the study more generalizable. Selection of interviewees in cases 1, 3, and 4 was required for the interviewees to operate within same context but sharing different responsibilities or being involved in different stages of a product's lifecycle that, presumably, affected their interest in SAV and desired level of abstraction of SAV. Interviewees from case 2 were system and design architects mostly from different projects in Ericsson that allows us to compare and contrast different applications of SAV and preferred abstraction level between different level architects based on different projects within same company. The 4th case, investigated by the authors of this paper, includes a software designer, a system manager, a system architect, a design architect, and a functional tester, all of whom are involved in the same project. The advantage of the final data set is access to data from 4 different cases, which were never analyzed as one before. Interviewing is also a lengthy process and it is difficult to obtain data from multiple cases in course of a semester that can be avoided by integrating newly conducted interviews (case 4) with previously conducted interviews (cases 1-3). Further, considering data from larger number of cases, provided by preexisting data set, builds external validity, by including cases of different backgrounds and development approaches. Lastly, analyzing a preexisting dataset may be viewed as an advantage, since possible perception based biases are eliminated.

The interview questions were divided into 4 categories:

1. Background questions
2. Software Design Process
3. Existing SAV of the system
4. Different levels of abstraction

Category 1 included questions about interviewee's position, department, and experience with SV techniques. Category 2 was applicable to stakeholders that were involved into

development process, and were asked to describe it in detail. Category 3 applied to all participants and consisted of questions about current ways a stakeholder used SAV to support his/her work and it which context it was done. It also contains questions that aim at obtaining data about what techniques, tools are being used, and what were the reasons for doing it. The last category applied to all participants and contained questions about comprehension of system at a different level of abstraction and needs for visualization at different levels of abstraction. Full list of questions is presented in Appendix A.

E. Data Analysis

Once data gathering was completed, case 4 interviews and 4 of case 3 interviews were transcribed. It was done in pairs to avoid misunderstanding over 3 weeks of time. Next, all 21 interviews were coded in order to condense the data. However, it is important to not excessively employ coding as it could "destroy the meaning" of data [12].

Coding was performed in 4 stages:

1. Open coding
2. Coding scheme composition
3. Second cycle coding
4. Tabular display of results

Open coding was conducted with an aim of identifying codes that could be used for second cycle coding. Then open codes were sorted to eliminate similar codes for the same data, and grouped by themes to produce a coding scheme. Once the scheme was completed, the interviews were coded again. Coding was done in a pair, first separately, and then cross-examining the results to see whether there are any considerable differences in how the interviews were coded. This was done to decrease the possibility of misunderstanding and tackle validity threats associated with this step, such as bias.

In order to avoid excessive coding and diminishing of data, produced coding scheme was rather simplistic and consisted of general codes such as:

1. Personal Information
    - 1.1. Name
    - 1.2. Stakeholder type
    - 1.3. Experience
    - 1.4. Responsibilities
2. Software Design Process
    - 2.1. Team Description
    - 2.2. Process Description
    - 2.3. Personal Involvement
3. Existing SAV Practices
    - 3.1. Demand
    - 3.2. Context
    - 3.3. Reasons for not using SAV (if applies)
    - 3.4. Information need
        - 3.4.1. Relationship
        - 3.4.2. Composition
        - 3.4.3. Complimentary
    - 3.5. Abstraction level
    - 3.6. Methods
    - 3.7. Tools
4. SAV practices improvement
    - 4.1. Lacking Information
    - 4.2. Other improvements

Gathered data about information need of different stakeholders' towards SAV was broad and requires further categorization. Three categories of information need were distinguished based on LaToza et al. [43], which included "Relationships", "Composition", and "Complementary" categories. "Composition" category included displaying static aspects of a system, such as its structural composition, such as method properties. "Relationships" category dealt with dynamic behavior of a system, rather than its composition, including control and data flows, and dependencies. Lastly, "Complementary" category included information related to change, such as history and intent of implementation, as well as other information needs that were not directly related to 2 previous categories, such as metrics.

LaToza et al. [43] concentrated on needs of developers, however, this categorization of information needs was general to be applied to other stakeholders as well.

Besides information need, techniques, and tools used by different stakeholders, level of abstraction is also a focus of this paper. Based on Gallagher et al. [7], three levels of abstraction are considered:

1. Low level, or code level, which is directly related to an "underlying artifact" ;
2. Medium level, which is problem specific level of visualization, such as sequence diagrams;
3. High level, or architectural level, which comprises overview of structure of an architecture and relevant metrics.

Based on this definition, levels of abstraction required for each stakeholder type was derived based on recorded data about level of detail and information need.

Additional data included stakeholder's experience, responsibilities, interests, improvements or limitations of current tools, and team composition, which could help motivating differences between different stakeholders or cases.

Then the condensed data was presented in a tabular form, with list of codes sorted from most to least important in a column on the left, and related quotations from each interview in columns on the right. This provided an effective scheme of data condensation for further sorting and result display.

Due to large amount of data, it needed to be categorized before it was to be analyzed. The main categories of data were stakeholder type, information need, techniques used, level of detail, type of tools used, level of demand, and reasons for not employing SAV, if it applies.

Quantitative data is minimal in this paper, only representing number of stakeholder exhibiting an interest in data that SAV displays, specific techniques, or levels of abstraction. This data could be converted into percentage, but considering, that there is only 21 interviews, it could be misleading.
As it is, numbering stakeholders interested in different aspects of SAV gives a general overview of their needs, displays patterns and correlations more efficiently. This gives "familiarity with data and preliminary theory generation" [12], and prompts viewing data from different perspective via employing "cross-case pattern search using divergent techniques" [12].

Lastly, after the interview results were discussed in respect to each other, they were also discussed in respect to literature review results, comparing it to complementing literature and contrasting with conflicting literature. This step does not only aim at answering the research questions, but also builds "internal validity, raises theoretical level, and sharpens generalizability" [12].

V. Literature Review

A. *Stakeholders*

A number of reviewed studies [2, 6, 7, 8, 5, 21, 23, 24, 33] from the field acknowledge the differences in requirements for SAV depending on a stakeholder, however, very few mention concrete techniques or levels of abstraction, appropriate for each stakeholder.

A list of stakeholders which may benefit from use of SAV differs from study to study as well. According to Mattila et al. [2], visualization is used mainly by developers, testers, architects and project managers. IEEE-1471 proposes four types of stakeholder, including users, acquirers, developers, and maintainers, while Gallager et al.[7] expands this list by adding architects, operators, testers, designers, development managers, sales and field support, and system administrators. Ghanam and Sheelagh [8] includes same stakeholders as Mattila, but noting that customers might be another stakeholder that would be interested in SAV. Both Panas et al. [21] and Priya et al. [24] propose developers, architects and project managers to be general stakeholders. Lastly, when reviewing stakeholder for SAV tools, Telea et al. [5] distinguishes three main stakeholders, which are technical users, project managers and consultants. Considering these examples, the most prominent stakeholders, which are included in all reviewed papers are developers, architects, and managers. These stakeholders encompass difference in demand for visualization techniques, and level of abstraction, and will be used as primary stakeholders in this paper.

According to Ghanam and and Carpendale [8] managers are interested in monitoring "progress of the project and determine the completion of the development goals". In addition, project managers could use visualization to determine what components of a system have high development or maintenance cost, as well solving problems related to resource management and meeting deadlines [21]. High-level visualization may help managers to understand the reasoning behind time estimates by developers and improve overall communication between different stakeholders [2, 6, 33]. Overall, in case of project managers, SAV should support monitoring of evolution of a system over extended period of time, providing information about general trends, such as "architectural erosion, rule violation, and quality decay" [5]. Considering that famously "20% of items that cause 80% of the problems can be solved by looking at distributions, not individual artifacts", project managers require high level of abstraction in conjunction with techniques that can simultaneously display numerous attributes or metrics, such as "treemaps and dense pixel charts"[5].
Architects, on the other hand, require lower level of visualization, displaying attributes of a designed architecture, such as complexity, coupling and cohesion [8]. Appropriate visualization can also aid identifying components for reuse [21], software architecture documentation [6, 22, 8], and monitoring software architecture evolution [6, 22, 2]. Overall, architects require visualizations that enable navigation of "software structure, dependencies, and attributes such as quality metrics [5]".
While managers approach SAV with aim of monitoring changes of the systems over time and completion of milestones, developers concentrate on code changes and its impact [8]. Generally, "developers require visual modelling support to help them effectively design and reason about the software components of complex applications" [35]. SAV aids developers and maintainers in system comprehension [8, 33, 6], and monitoring recent changes [8] while testers can be helped by SAV when exploring code for anomalies [33].
According to Telea et al.[5], stakeholders concerned with low level of abstraction, such as developers, maintainers, and testers, are interested in similar techniques as architects, such as treemap techniques, and hierarchically bundled edges, that produce "readable, clutter-free layouts of thousands of entities and relationships with zero user intervention" [5].
However, regardless of benefits of employing SAV being demonstrated by numerous studies, which are reviewed in the next section, developers and other low-level stakeholder are still not commonly adopting SAV to support their work [33]. Telea et al. [5] claims that needs of developers and architects are satisfied the most comparing to other stakeholders, such as managers and consultants. Gallagher et al. [7] complements this view, claiming that majority of SAV tools cater to the needs of developers and maintainers, and thus "has been largely concerned with representing static and dynamic aspects of software at the code level" [7]. Marino et al. [33], on the other hand, claims that "developers have little support for adopting a proper visualization for their needs". Numerous tools and techniques are proposed with an aim of aiding developers [7, 5, 33], however, these "efforts in software visualization are out of touch with the needs of developers" [33] and developers are simply "unaware of existing visualization techniques to adopt for their particular needs"

[33]. LaToza and Myers [48 from 33] problem domains that developers deal with into three categories: “changes”, “element”, and “element relationships”. While developers are mostly concerned about “changes, “existing visualizations distribute their attentions among all three categories”. As a result, some problem domains that are particularly important for the developers, such as rationale, intent, implementation and refactoring, are lacking support, while other problem domains, such as history, performance, concurrency and dependencies, are well-supported.

Filtering visualization in order to display software architecture entities that a stakeholder is interested in at an appropriate level of detail is a process of abstraction. Gallagher et al. [7] distinguishes three level of program visualization based on level of abstraction: source code level, middle level and architecture level. Source code level visualizations are typically “low level” and relate directly to the “underlying artifact”. Middle level visualizations are “problem-specific” are aim to visualize problem area, that might include “sequence diagrams, abstract syntax trees (AST), dominance tree, concept lattices, control and data flow graphs“. Architecture level is abstract architecture visualization that aims to communicate design decisions and overall structure. In combination with metrics, architecture visualizations may satisfy needs of various stakeholders, such as visualizations of most costly components for managers, or design erosion visualizations for code designers.

B. Purposes and Benefits of SAV application

Most common categorization of SAV use cases are by architecting activities [6], problem domains [33], and purposes [6, 2, 7, 5, 25]. According to [45], “architecting is a process of conceiving, defining, expressing, documenting, communicating, certifying proper implementation of, maintaining and improving an architecture throughout a system’s life cycle”. Telea et al. [5] noted that SAV techniques “can be used to support any stage of the software architecting process, i.e., analyzing, synthesizing, evaluating, implementing and evolving architecture”, while Li et al. [FROM 6] defines architecting activities to be architecture recovery, architectural evolution, architectural evaluation, change impact analysis, architectural analysis, architectural synthesis architectural implementation, and architecture reuse. Shahin et al. [6] conducted a systematic literature review, and determined, that 47% of reviewed studies were activity to use SAV most frequently. To the large extent, SAV also supports architectural evolution dedicated to SAV application within context of architecture recovery, making it the architecting (30%), architectural evaluation (20%), change impact analysis (18%), and architectural analysis (18%). Less supported activities, according to Shahin et al. [6] were architectural synthesis, architectural implementation, and architectural reuse.

LaToza et al. [43] categorized “hard-to-answer” questions about code into categories, such as questions about changes (debugging, implementing, policies, rationale, history, implications, refactoring, testing, building and branching, and teammates), questions about elements (intent and implementation, method properties, location, performance, concurrency), element relationships (contracts, control flow, dependencies, data flow, type relationships, and architecture). Problem domains of rational, intent and implementation, debugging, refactoring, and history were distinguished as most frequently asked questions categories from developers’ point of view. Addressing this problem domains could be aided with SAV tools and techniques, however, according to Marino et al. [33] some of the most relevant problem domains are least supported, such as rationale and refactoring, while least relevant domains, such as dependencies and concurrency, and are supported to a far larger extent.

Shahin et al. [6] reviewed related studies published between 1999 and 2011, and divided purposes of using SAV techniques into 10 categories from most to least frequent. Improving understanding of architecture evolution is the most frequent context of using SAV with 26% of reviewed papers reporting it. Improving understanding of static characteristics of architecture and improving search, navigation and exploration of architecture design are following with 24% of studies. 21% of papers studied SAV application within context of improving understanding of architecture design through design decisions visualization. Less frequent purposes of SAV employment are supporting architecture re-engineering and reverse engineering (13%), detecting violations, flaws, and faults in architecture design (11%), provide traceability between architectural entities and software artifacts (11%), improve understanding of behavioral characteristics or architecture (6%), checking compatibility and synchronization between architecture design and implementation (6%), and supporting model-driven development using architecture design (2%).

Besides Shahin [6], other numerous papers study SAV application with purposes of system and code comprehension, especially in context of software evolution. According to Sharafi [17], “from 50% to 75% of the overall cost of the system is dedicated to is maintenance”, while “during maintenance developers spend at least half their time reading system source code in order to understand it”. Similarly, Telea et al. [5] claims that “software maintenance costs about 80% of a software product’s total life-cycle costs, and 40 % of that cost is software understanding”. Chikofsky and Cross [22] supports these claims, stating that cost of maintenance ranges from 50% to 90% of costs of software total life-cycle. The authors add that “the cost of understanding software, while rarely seen as a direct cost, is nonetheless very real” and ”it is manifested in the time required to comprehend software, which includes the time lost to misunderstanding”. Additionally, Chikofsky and Cross [22] expresses a view, that “graphical representation have long been accepted as comprehension aids”, that was supported by other numerous papers [2, 6, 5, 25, 31, 32, 33].

Further, SAV is frequently mentioned within context of reverse engineering. According to Chikofsky and Cross [22], its purpose is to "increase the overall comprehension of the system for both maintenance and new development" that can be done via generation of alternate views; while according to Shanin [6], SAV "represents its software components and the relationship between those components at different levels of abstraction" within context of reverse engineering. Redocumentation, as a part of reverse engineering, can also be aided by SAV and is defined as "creation or revision of a semantically equivalent representation within the same relative abstraction level" [22]. Mattila et al. [2], Telea et al. [5], and Balzer [25] also mention SAV within context of reverse engineering.

Considering that system's implementation evolves over time, its "architecture design and implementation may not be compatible" [6]. Architecture erosion, "as-implemented and as-planned" architecture can be displayed and monitored with aid of SAV, as well as identifying architectural violations [2, 6, 7, 5, 31, 22].

Besides maintenance, reverse engineering, and comprehension, SAV supports "collaboration and engagement, optimization, assessment and comparison" [2], "highlighting architectural patterns or patterns extracted from code bases, assessing architecture quality" [5], as well as "providing guidance to software life cycle" [32]. Employment of SAV to support management task and communication was also mentioned in a number of studies [2, 5, 31], however, Shanin et al. [6] noted that visualization is infrequently used to aid management in comparison to other problem domains.

C. SAV Techniques

According to Koschke [37], "visualization techniques are widely considered to be important for understanding large scale software systems". However, "knowing what to visualize and how to present information are themselves daunting issues" [21]. Not all visualizations are appropriate for a given problem domains, information need of a user, or level of abstraction. Many SAV techniques are inappropriate for displaying diagrams generated from large code bases with high number of entities. When employing an inappropriate technique, there is a risk of displaying too much information that would be difficult to comprehend even in a graphical representation that is rooted in "visual complexity associated with the limitations of human brain capabilities and short term memory capacity" [8]. Samia and Leuschel [30] reinforce this view, stating that "visualizing large amount of information as a graph can be ineffective, even though it is accurate". Therefore, it is vital to determine what is the user's information need, required level of abstraction and detail, and a problem domain that visualization targets. Furthermore, different techniques might require different level of tool support. Whether some high level abstract diagrams might be drawn manually, some low level diagrams, such as node-to-link, require fully automated tools.

One of the most common categorization of SAV techniques is static versus dynamic visualization. Both Gallagher et al. [7] and Grundy and Hosking [35] advocate for usage of both dynamic and static visualizations during design and development. According to Gallagher et al. [7], static representations visualizes "information which can be extracted before runtime, for example, source code, test plans, data dictionaries, and other documentation", while dynamic representations display system's behavior during runtime, that is most appropriate for "relationships between components of a system that will be formed only during execution due the nature of late-binding mechanisms such as inheritance and polymorphism". Static visualizations can provide information regarding overall structure of a system at different levels of abstraction to cater to various stakeholders' needs. Dynamic visualizations, on the other hand, are particularly relevant to developers' needs, aiding understanding of system's correctness and high-level behavioral characteristics that cannot be otherwise determined from static representations [35]. Ideally, in order to achieve effective navigation between static and dynamic representations, visualization structures should be consistent [35]. According to Grundy and Hosking [35], many visualization tools support separate dynamic and static representations, but lack common visualization methods, such as "modelling languages or views, and are thus difficult to formulate and interpret".

Another approach to categorization of SAV techniques is described by Priya e al. [24] and Ghanam and Carpendale [8] and includes multiplicity of view, dimensionality and metaphor. Multiplicity of view is one of the most common concepts within SAV, being mentioned in 52% of studies related to SAV and being capable of supporting many software engineering activities, except for requirements engineering [2]. Ghanam and Carpendale [8] account two "schools of thought" regarding multiplicity of view: first, that visualization should contain a number of different views in order to satisfy different audiences depending on required level of abstraction; and second, that single view, carefully designed, may provide information more effectively. Multiple view caters to individual needs of stakeholders, playing on the difference between them, while single view underlines common purpose of visualization, "enhances communication between the different stakeholders by allowing them to reach a common understanding of the architecture" [8]. Panas et al. [21] argues for use of single view visualizations, stating that even though multiple view visualization are still widely accepted, it disturbs communication between different stakeholders as they refer to different visualizations and data, difficult to navigate, and harms "mental picture" of system's architecture in user's mind [21]. Further, multiple views produce large volumes of different data that are difficult to manage and store [21].

In SAV, dimensionality refers to distinction of visualization in 2D or 3D. Visualizations in 3D can be advantageous when it comes to representing and comparing metrics of various

components, while attempts to visualize some metrics in form of gradient or transparency in 2D failed to increase comprehension [8]. Additionally, 3D visualization attracted a lot of attention of the research community which reasons that "only two dimensions to represent highly dimensional data can be too overwhelming for the viewer to comprehend" [8]. Despite this advantages, a number of papers criticise 3D visualization technique. Ghanam and Carpendale [8] argues that "a carefully designed 2D representation of an architecture should be capable of representing more than two dimensions in the dataset". Wettel and Lanza [19] states that 3D SAV is not widely recognized due to issues with navigation and interaction, lacking locality and casing disorientation. According to Priya et al. [24] "this trend [of 3D visualizations] has been most probably supported by the advancement in related graphic technologies (software and hardware) rather than empirical evidence of the advantages of using real metaphor in software visualization". Ghanam and Carpendale [8] shares this view, stating that there is no concrete evidence that an added dimension can aid comprehension better than 2D visualization.

Both Ghanam and Carpendale [8] and Wetter and Lanza [19] propose using 3D visualization in conjunction with metaphor-based visualizations, which "allows the viewer to embed the represented elements into familiar context, thus contrasting disorientation".
According to Shahin [6], metaphor-based visualization refers to using familiar real-world objects to visualize architecture, like cities, which makes it particularly intuitive and reduce visual complexity. Carpendale and Ghanam [8] define metaphor-based visualization as mapping SA and metrics to metaphors, be it geometrical shapes, or real metaphors such as buildings, and state that this method can provide a user with more intuitive understanding of architecture. Kobayashi et al. [26] shares the same view, stating that "a city metaphor is widely adopted in many studies, it is intuitive and navigable, and it can represent various software structures and metrics at the same time".

Merino et al. [33] divides visualization techniques into two different types: techniques, using geometric transformation, that "explore structure and distribution" and pixel-oriented techniques that are capable of representing large amount of data. [25] Geometrically transformed visualizations are "frequent because node-link techniques that belong to this category are profusely used by visualizations that explore relationships", while Dense Pixel techniques are popular because they "contain techniques suitable for depicting massive data sets".

Lastly, Shahin et al. [6] identifies four primary types of SAV techniques that are: graph-, notation-, matrix-, and metaphor-based visualizations. Graph-based visualization uses "nodes and links to represent the structural relationship between architecture elements and it puts more emphasis on the overall properties of a structure then the types of nodes". Graph-based technique attracted the most of researchers attention in comparison to other techniques, being reported in 49% of reviewed literature, as well as being most frequently employed technique in the industry due to its capability to visualize "overall properties of a structure, which is useful for all types of projects to get an overview of the architecture" [6]. This technique category is the most supported by automatic tools, since it requires to be generated from the code. Examples of graph techniques are hierarchical edge bundles [39] and clustered graph layout [40], displayed in fig 1 and 2

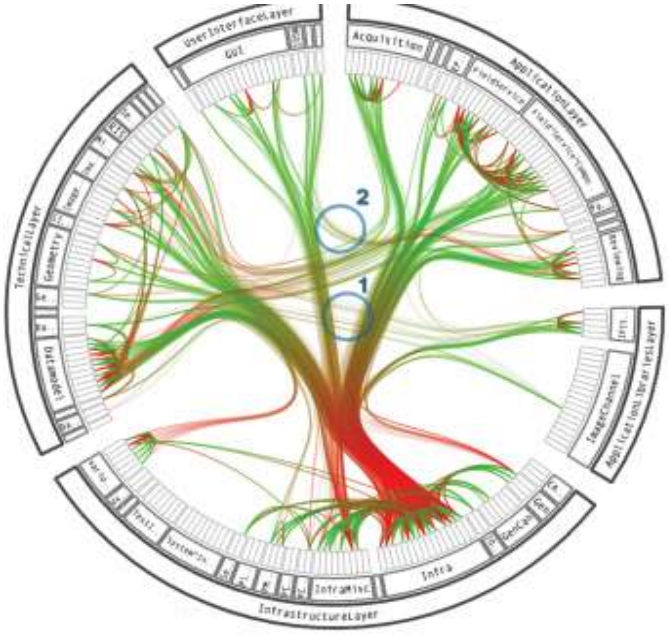

Figure 1. Hierarchical edge bundles [39]

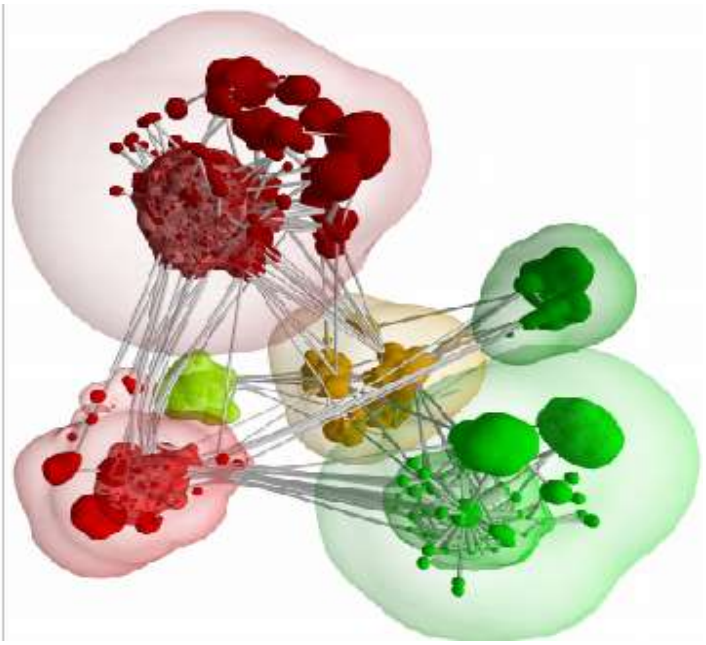

Figure 2. Clustered graph layout [40]

respectively.

Hierarchical edge bundle technique in figure 1 represents nodes as segments of inner circle that are part of abstracted layers. Links represent calls from a node to a node, with callers in green and calee in red. This visualization can also be adjusted in accordance to required level of detail, providing both low level and high level information and thus catering to various stakeholders' needs. Similarly, clustered graph layout in Figure 2, is an abstract visualization of clusters of edges or parent edges that can be adjusted in level of detail to suit user's information need.

However, this techniques can produce large and difficult to read graphs, with cluttered and omitted edges due to "high interconnectivity between the large amount of components" [38]. This disadvantage can be addressed by employing matrix-based visualization, a complementary to graph-based visualization, which is capable of displaying structural information about a large system. However, it proves to be a difficult to keep a mind map of a system's hierarchy, and it is less intuitive than other visualization techniques [38, 6]. Lungu and Lanza [41] present semantic dependency matrix for "displaying details about dependency between two modules which groups together classes with similar behavior" and edge evolution filmtrip in figure 4, which visualizes "the evolution of an inter-module relation through multiple versions of the system", with examples of both displayed in figure 3 and 4 respectively.

Another common technique category is notation-based techniques, consisting of SysML, UML and other specifically designed custom modelling and visualization notation-based techniques [6]. According to Shahin et al. [6], 41% of reviewed studies focused on notation-based visualization, while 81% of notation-based SAV related studies were published in last 5 years [2009-2014], signifying increase in interest in this technique. Notation-based visualization is second most frequently mentioned technique in related studies (after graph-based) [6], and also became an industrial standard [38]. According to Balzer et al. [25], Unified Modelling Language (UML) is the most widely employed modelling language, in which class diagrams are used to model "static structure of the system", that can be grouped into packages and thus adjust level of abstraction. Khan et al. [38] states that UML was firstly developed to display inter-class relationships, portraying composition, aggregation, generalization, and inheritance. Grundy and Hosking [35] mirrors this sentiment, stating that UML sufficiently supports lower-level visualizations, but adds, that it is limited when it comes to displaying high-level views of architecture, considering that deployment diagram, showing "machine and process assignment and interconnection", is the only option of displaying high-level view of architecture. Balzer et al. [25] states that UML notation do not include "advanced graphics and visualization techniques" and prompts users draw diagrams themselves, that, in turn, " decreases information density and control over the level of abstraction, which limits scalability"[25]. Shahin et al. [6], on the other hand, states that notation-based visualization are second best when it comes to tool coverage (again, after graph-based), with semi-automatic and automatic tools, however, Shahin's work overviews

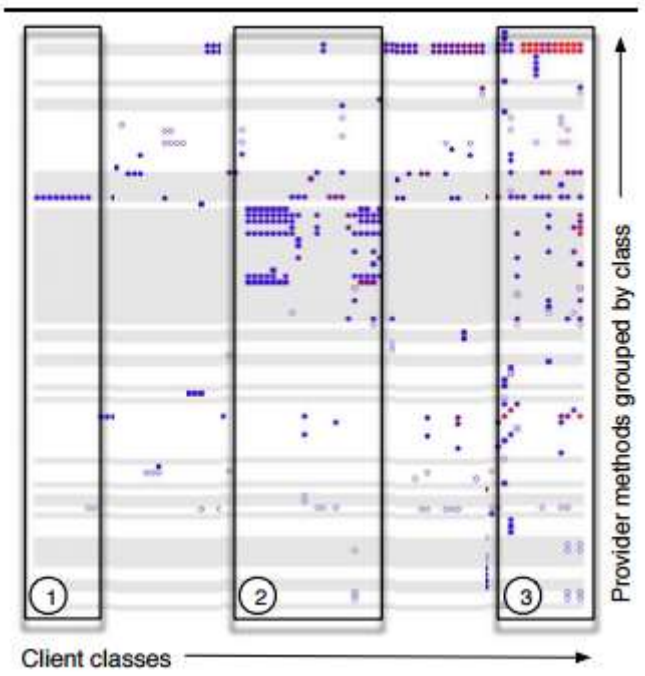

Figure 3. Semantic Dependency matrix for dependency between 2 modules [41]

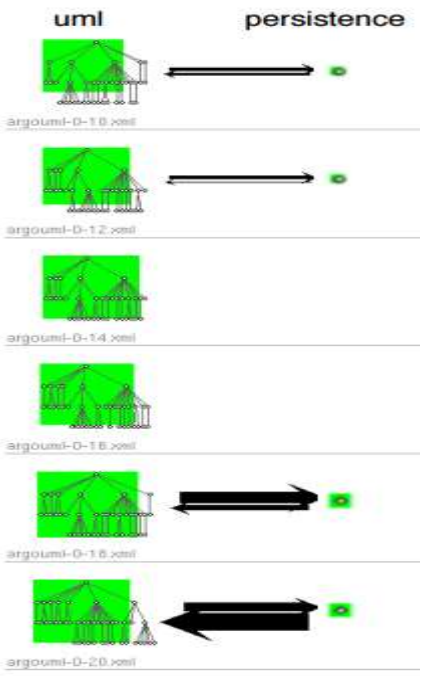

Figure 4. An example of edge evolution Filmstip [41]

scientific studies, and not SAV employment in the industrial context, which could explain the contradiction. Khan et al. [38] argues that generating UML diagrams from a large codebase can lead to information overload due to 'the amount of textual information depicted by each component", and adds that "these graphs grow exponentially with each additional component" added.

Previously mentioned metaphor-based visualization are the least frequently mentioned in studies (13%), according to Shahin et al. [6]. However, in recent years, an interest to metaphor-based visualizations grew, with various new tools being proposed, an example of which is Vizz3D tool by Panas et al [21]. The tool presents an architecture in form of a city, using metaphors such as buildings, textures, cities, pillars,

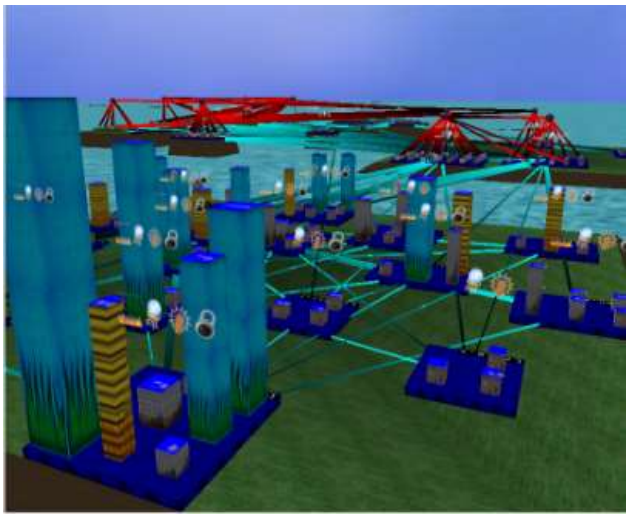

Figure 5. Vizz3D visualization of C++ program Architecture [21]

water towers and landscapes representing functions, source code metrics, source files, header files, and directories respectively. The generated visualization (Fig. 5) is

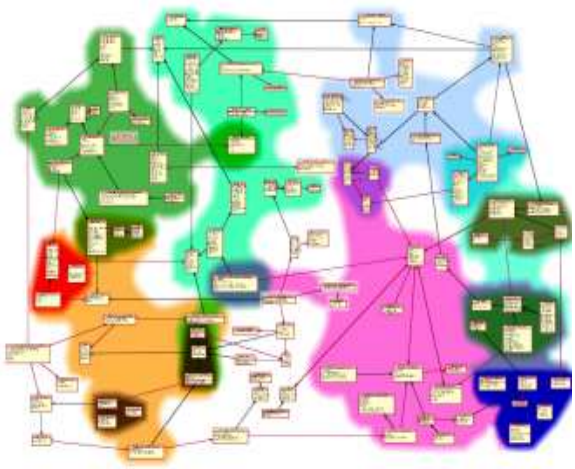

Figure 6. Generated UML model with 12 areas of interest [20]

predictable and keeps to a same layout patterns when run multiple number of times which allows a user's maintain a common, unchanged mind map of the system. Generated visualization is capable of displaying software complexity information, oversized functions, unsafe functions and run-time information.

A number of studies also employ different techniques such as UML or metaphor-based embedded with visualization of metrics or areas of interest, such as "design complexity, resource usage, system stability" [38], "performance, trust, reliability, or structural attributes, correspond to the system architecture" [12], that are vital to understanding of complex software systems, according to Byelas and Telea [20]. Wettel and Lanza [19] use metaphor-based approach, while "mapping source code metrics onto size and type of building", color and transparency in CodeCity tool. In figure 6, Byelas and Telea [20] visualize architecture in conjunction with areas of interest, such as performance, structural attributes, and reliability, by grouping components by these properties and coloring the encircled components' area. Another tool, combining UML and metrics is Metric View [42], which is capable of visualizing metrics such as system cohesiveness, quality, and component coupling, by adding metric icons on each UML component.

D. Tools

Most of the studies (92%) reviewed in Shahin et al. [6] included descriptions of, or proposed, a new visualization tool, which signifies that tool support is a major concern for researchers and practitioners. Further, 42% of proposed tools were automatic, 47% were semi-automatic, and 11% were manual. However, according to Merino et al. [33] even though many tools are being proposed within research community, "few prototypes were maintained and extended over time", with average lifespan of a tool being about 3.7 years.

Satisfying all stakeholder requirements remains to be a problem as well. According to Gallagher et al. [6], none of the reviewed tools supported all stakeholders' demands for SAV and thus, for a complete visualization, a team should use a combination of tools, which, in turn, could be complicated. However, it is unclear whether an "ideal" tool would be possible to implement or whether it would even be desirable, since there can be "a risk of introducing cognitive overload to some stakeholders in the architecture". The authors then concluded: "It may be that one-fits-all-approach may increase information overload and that a collection of small tools appropriate to each stakeholder's task may be preferable".
However, adoption of a new visualization tool can also prove to be problematic. According to Telea et al. [5], while observing adoption of new tools, the researchers met with "moderate to strong skepticism regarding innovative AVTs [architecture visualization tools]", while discerning "significantly reduced understanding for time and cost and improved results quality when projects that used no

visualizations adopted AVT" or "replaced an existing tool with a better one".

VI. RESULTS

This section presents coding results, organized by its relation to research questions. Summary of each research question-related subsection is presented by the end of the subsections and denoted by boarders. Additionally, summary of interview coding results can be found in tables 1-7 on pages 21-25. Tables 1-3 present the results sorted by company or case for easier comparison of different stakeholders within same case, while tables 4-7 present same results, but sorted by stakeholder type, for easier comparison of same stakeholders from different companies. Lastly, table 8 on page 26 presents most common information needs, techniques, tools and level of abstraction, required by stakeholders.

RQ1: What is the current demand for SAV in the industry depending on a stakeholder?
Results for this sub-sections mostly comprise stakeholders' explicit statements regarding how useful SAV is or can be to support their work.

Three out of four developers from cases 1 and 3 responded that visualization is useful to some extent when it comes to understanding of architecture and communication. These developers stated, that "It could helpful while discussing architecture", and that "for a new developer coming in, it would be beneficial to have something", while it is being automatically generated. Fourth developer, in contrast, stated that it is definitely useful to support his work.

Design architects' responses included "very useful" and "useful" for understanding of architecture in cases 2 and 3; "sometimes" for tracking dependencies and understanding architecture in case 2, and "depends" on whether it is automatically generated, which would be favorable.

Responses of system architects were more affirmative, including "definitely useful" from two architects in case 3 and one in case 4; "useful" in case 1; and "somewhat useful" in case 2. Purposes of visualization for this stakeholder included "communicating vision of architecture", "overview of the system", "explaining architecture to other projects and non-technical stakeholders", "decision-making", and "communicating within a team"

Two managers from cases 4 and 3 found visualization useful when communicating, making decisions and understanding architecture. Another manager from case 3 implied that SAV is useful when communicating as well.

Test engineers from cases 1 and 3 found visualization useful if it is complemented with metrics. Case 4 function tester stated that it can be very helpful for other stakeholders, such as developers and architects, however, it is of limited use.

To summarize, based on this data, system architects found visualization most useful followed by managers. Design architects viewed visualization as mostly useful; while developers responded that it aids communication and introduction of new developers, and is useful if automatically generated

RQ2: What is the information need of different stakeholders towards SAV?

Stakeholders' information needs were divided into 3 categories, based on LaRoza et al. [43]:

1. Relationships, concerning visualization of relationships between different software entities at different level of detail and includes dependencies, control and data flow, i.e. dynamic aspects of the software.
2. Composition, concerning structural composition at different levels of detail, concerning intent, implementation, and method properties, i.e. static aspects of software.
3. Complementary, which includes additional information that is not directly related to entities or relationships between them, such as metrics, corresponding requirements, history of change and authors, and implications of new flows.

Information needs in Relationships category
Figure 7 presents a unified view on stakeholder needs in relationships category for all 4 industrial cases, with stakeholders from Volvo (case 1) colored green, Ericsson (case 2) colored purple, Tetra Pak (case 3) colored yellow, and Ericsson (case 4) colored blue. Middle column includes entities, dependencies between which are information need for the stakeholders. Figure 8 and 9, share the same data, but split into 2, including data from Volvo and Ericsson, and Tetra Pak and Ericsson respectively, to improve readability.
Based on figures 7 and 9, comparing stakeholders' needs from cases 3 and 4, System developer in case 3 is interested to see relationships between classes and packages, while software developer is interested in relationships between classes, packages, and layers. Design architect is case 3 is interested in relationships between classes, clusters of classes, and components, while design architect in case 4, is limited to components only. System architect in case 3 is interested in seeing relationships between clusters of classes, modules, and components, while system architect in case 4 is interested in modules, layers, components and systems. Additionally, in case 3, one of developers is not using SAV to support his work, as well as test engineer. Function tester in case 4 is interested in relationships between components, while Management from both cases require information about relationships of systems, subsystems and, in one case, components.

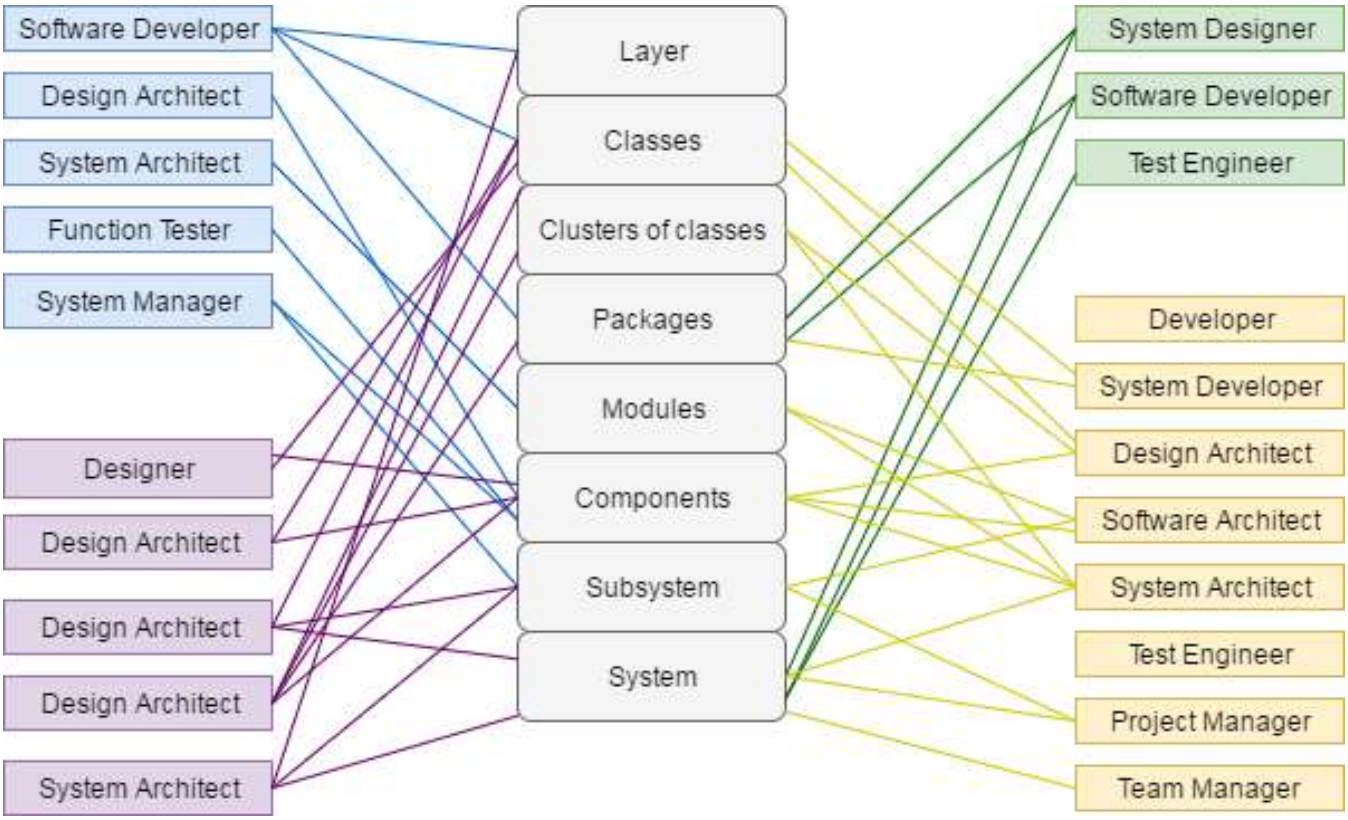

Figure 7. Information need in relationships category for cases 1-4.

In case 1, both system designer and software developer are interested in relationships between software compositions (SWC) and Electrical Control Units (ECUs), which in this diagram are denoted as packages and systems respectively.

In case 2, members of the same stakeholder group show different interests, for example, 1st design architect is concerned with relationships between classes and components; 2nd design architect is interested in relationships between classes, subsystems, and systems; while 3rd design architect required information about relationships between classes, clusters of classes, and components. Designer is concerned with relationships between classes and components, and system architect is interested in viewing relationships between subsystems and systems.

Based on figure 7, dependencies between components are the most demanded, being mentioned by 10 stakeholders. Next is dependency between systems, required by 9 stakeholders. Dependency between classes is important to 6 stakeholders, packages and subsystems were mentioned by 5 stakeholders each. Lastly, relationships between modules and layers had lowest demand, being mentioned only 3 times each.

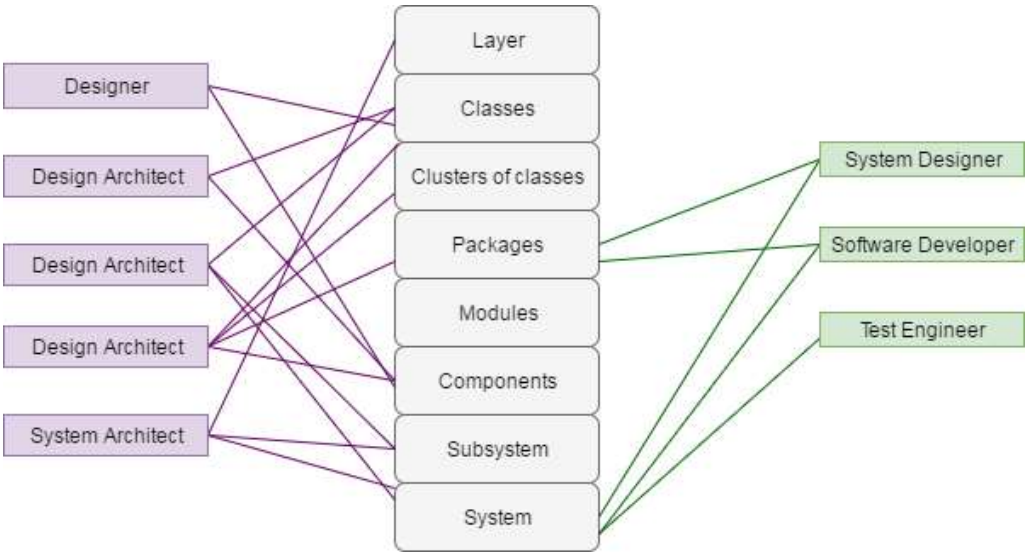

Figure 8. Information need in relationships category for cases 1 and 2.

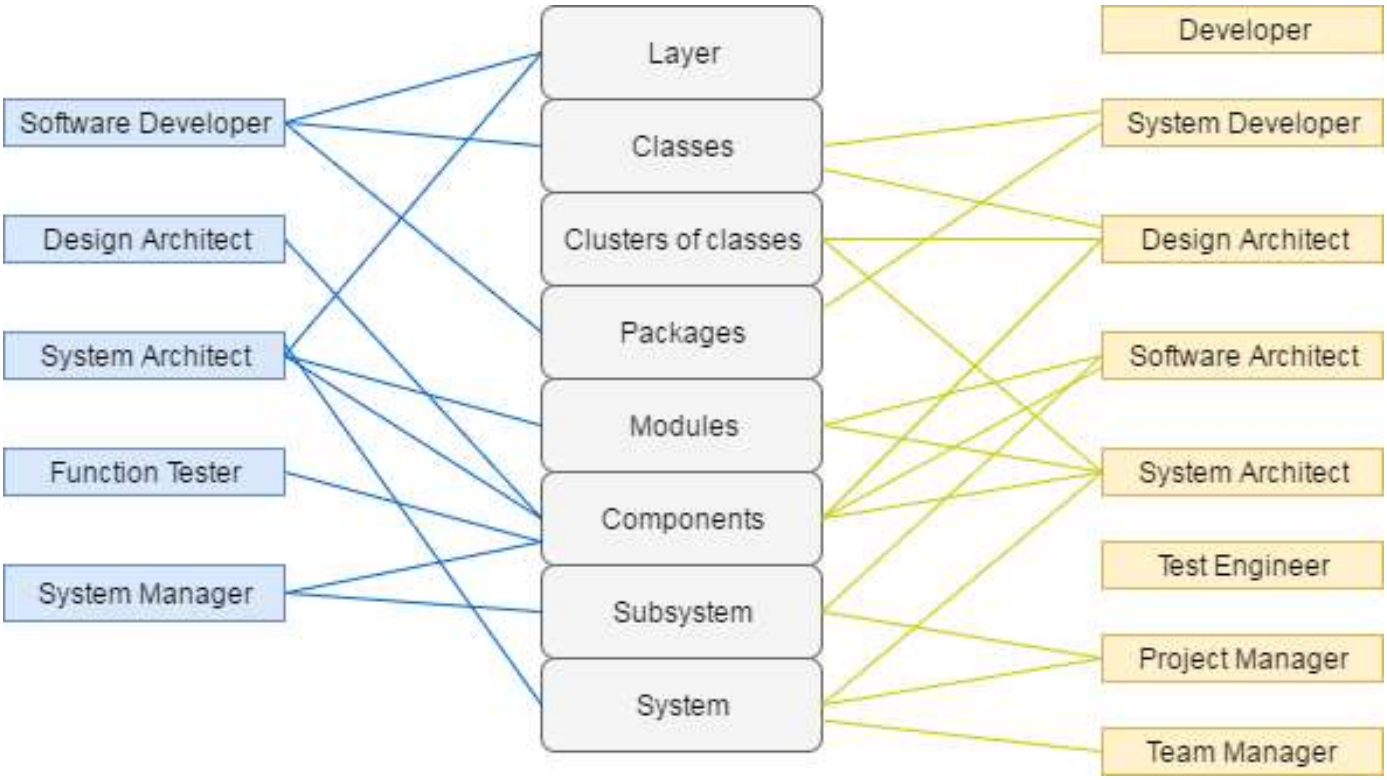

Figure 9. Information need in relationships category for cases 3 and 4.

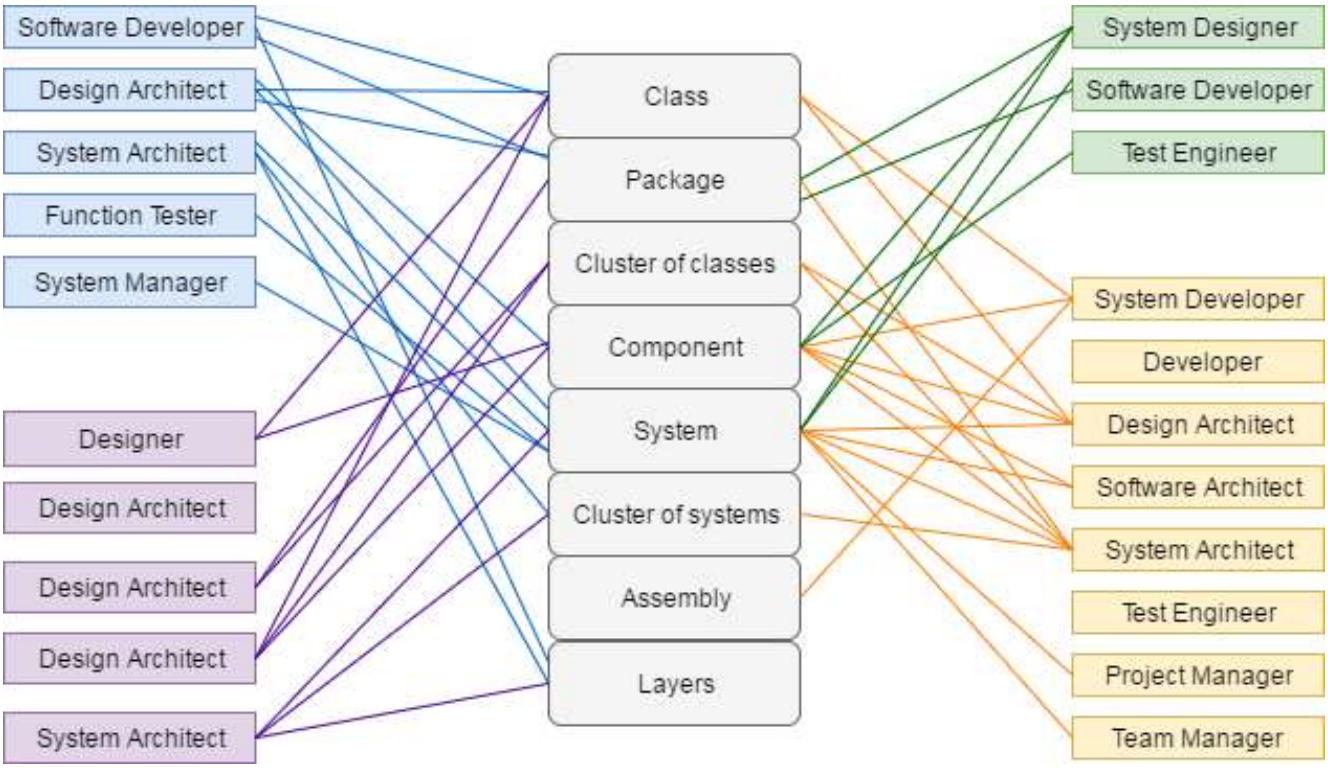

Figure 10. Information need in composition category for case 1-4.

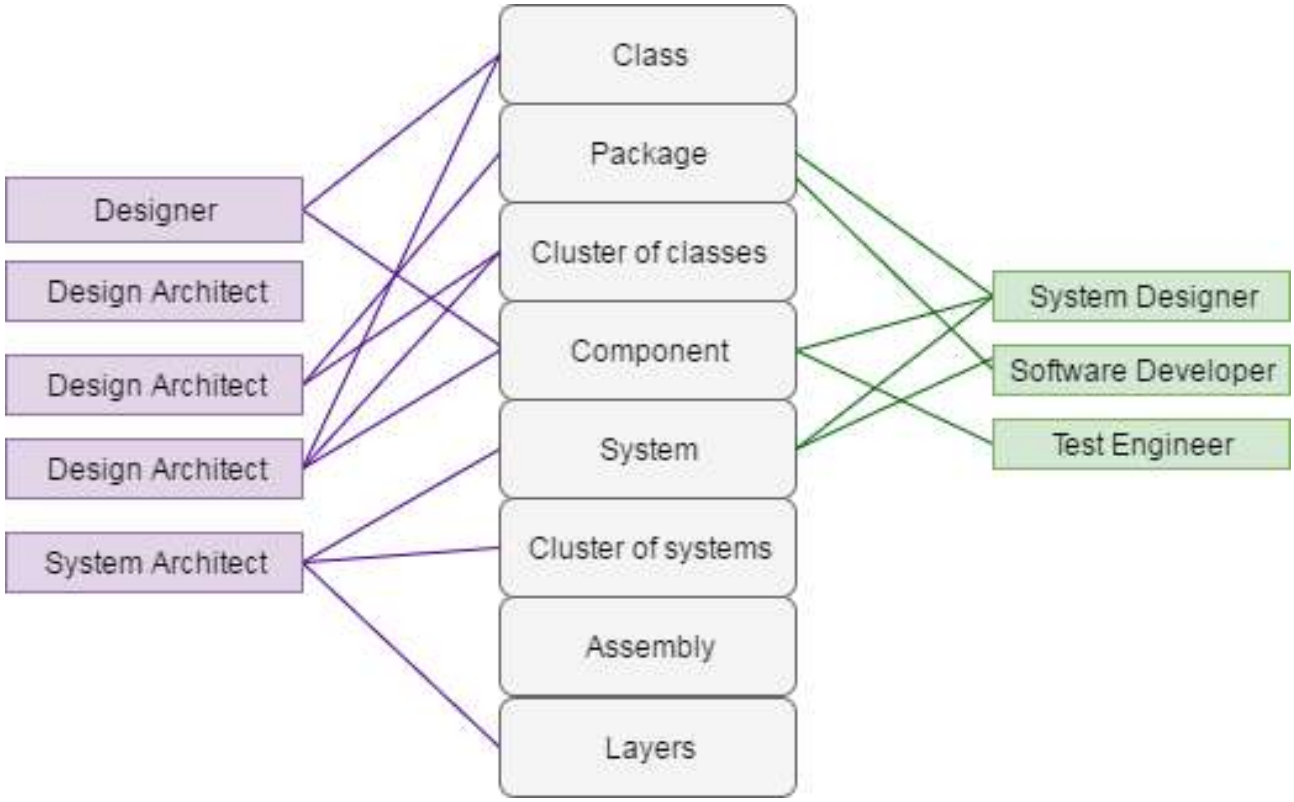

Figure 11. Information need in composition category for cases 1 and 2

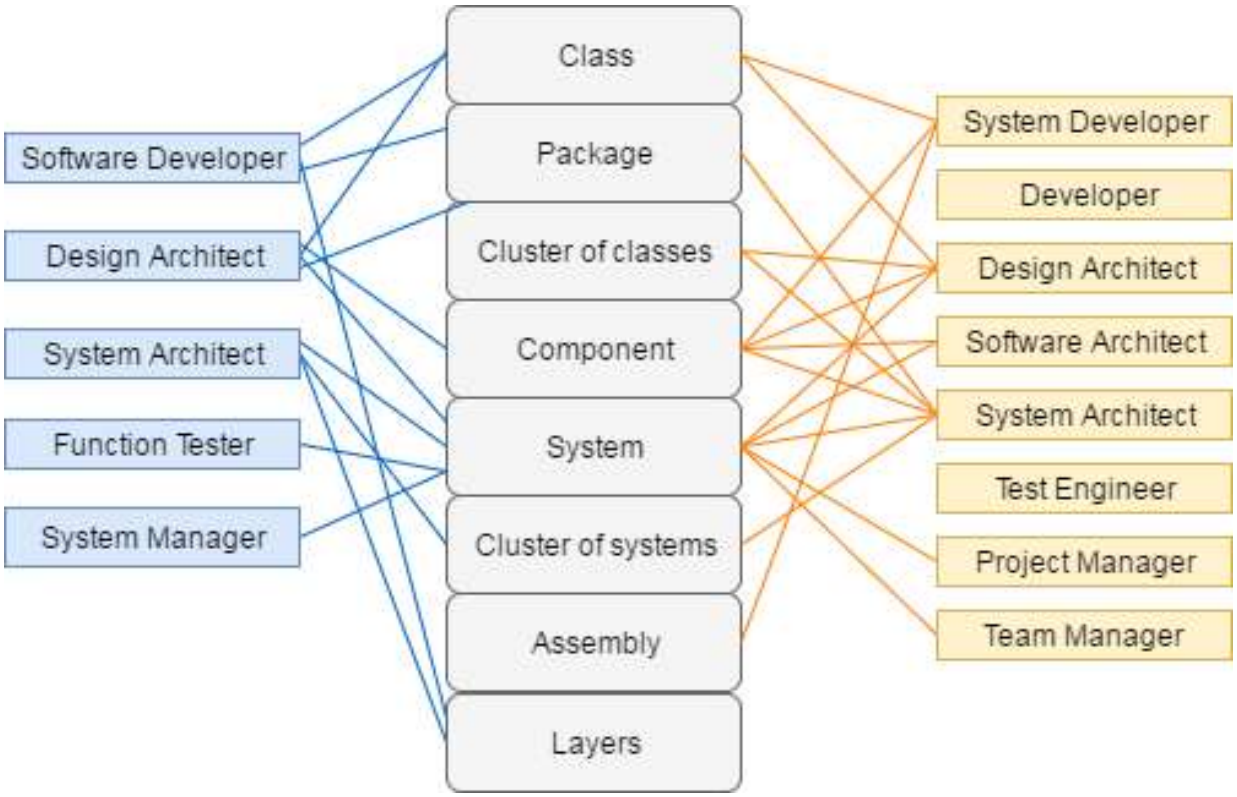

Figure 12. Information need in composition category for cases 3 and 4.

Information need in Composition category

Figures 10-12 display stakeholders' needs when it comes to composition of different software entities, which are listed in a middle column. Similarly to figure 7, figure 10 presents a unified view on stakeholder needs for all 4 industrial cases, with stakeholders from Volvo (case 1) colored green, Ericsson (case 2) colored purple, Tetra Pak (case 3) colored yellow, and Ericsson (case 4) colored blue. Figure 11 and 12 show same data, but divided, displaying 2 cases each, cases 1 and 2, and cases 3 and 4 respectively. According to figure 12, a developer in case 4 is interested in composition of classes, packages, and layers, while one of developers from case 3 is interested in classes, components, and assemblies, and another developer did not use any visualization. Design architect in case 4 is interested in composition of classes, packages, components, and systems, while design architect in case 3 is concerned with composition of classes, clusters of classes, components, and systems. System architect in case 3 is interested in composition of systems, clusters of systems and layers, while same stakeholder in case 3 is interested in composition of packages, clusters of classes, components and clusters of systems. System manager in case 4 and project manager and team manager in case 3 are all interested in system composition only.

In regards to figure 11, system designer in case 1 is concerned with composition of packages (SWCs), components (LACs), and systems (ECUs), while designer in case 2 in concerned with classes and components. One of the design architects in case 2 is not interested in composition, requiring dynamic behavior visualizations only, which are expressed in dependencies category. From other 2 design architects from case 2, one is interested in composition of packages and clusters of classes, and another in classes, clusters of classes and components. System architects in case 2 are interested in composition of systems, clusters of systems, and layers. In case 1, software developer requires visualization of package,

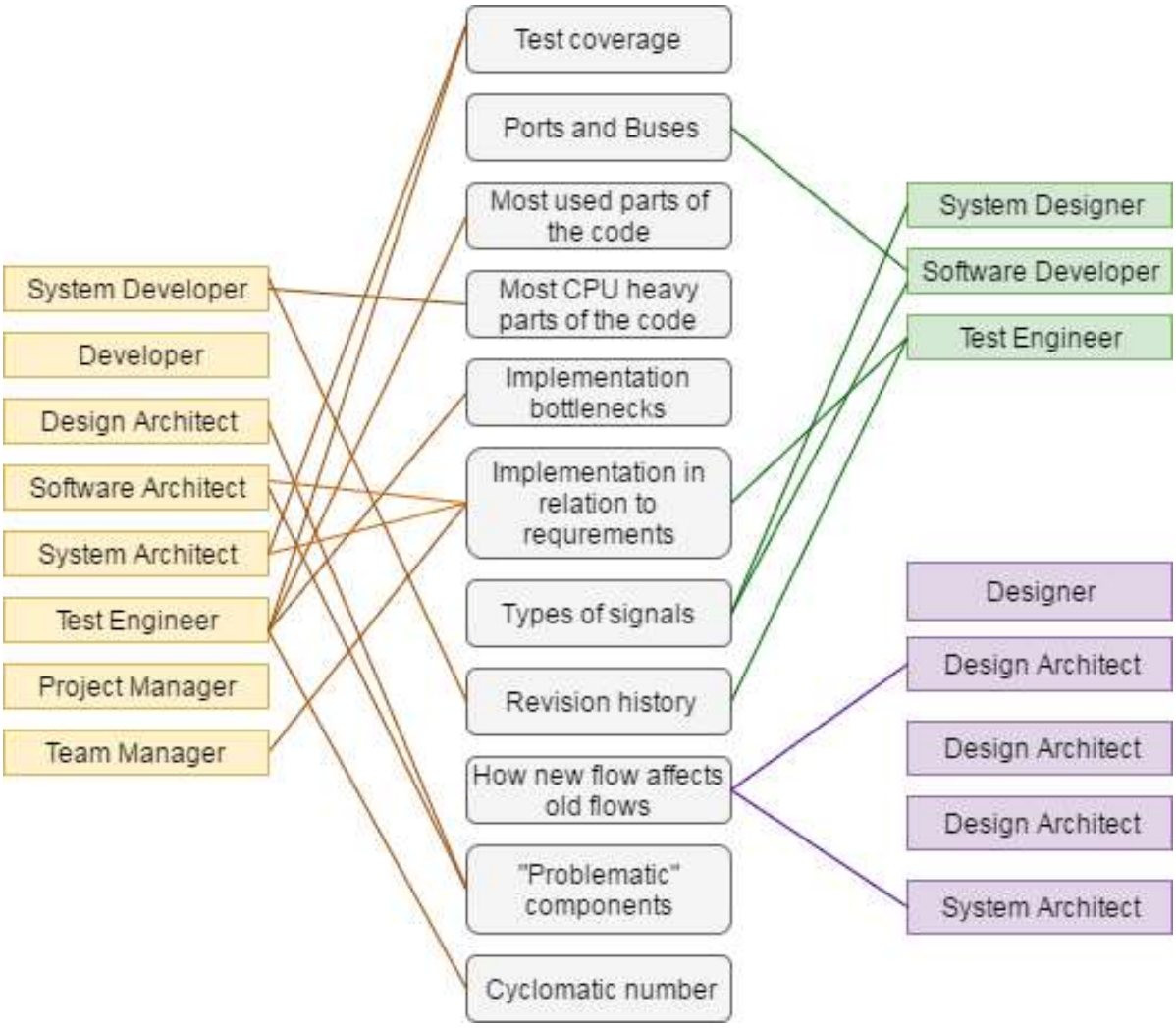

Figure 13. Information need in Complementary category for cases 1-4.

package and system visualization, while test engineer requires component composition visualization.

Based on figure 10, Component and system composition are most required, being mentioned by 9 stakeholders each. Next is class composition, mentioned by 7 stakeholders, and package composition, mentioned by 6. Cluster of classes was mentioned by 4, clusters of systems and layers by 3 each, and assembly was the least frequent, mentioned by one developer only.

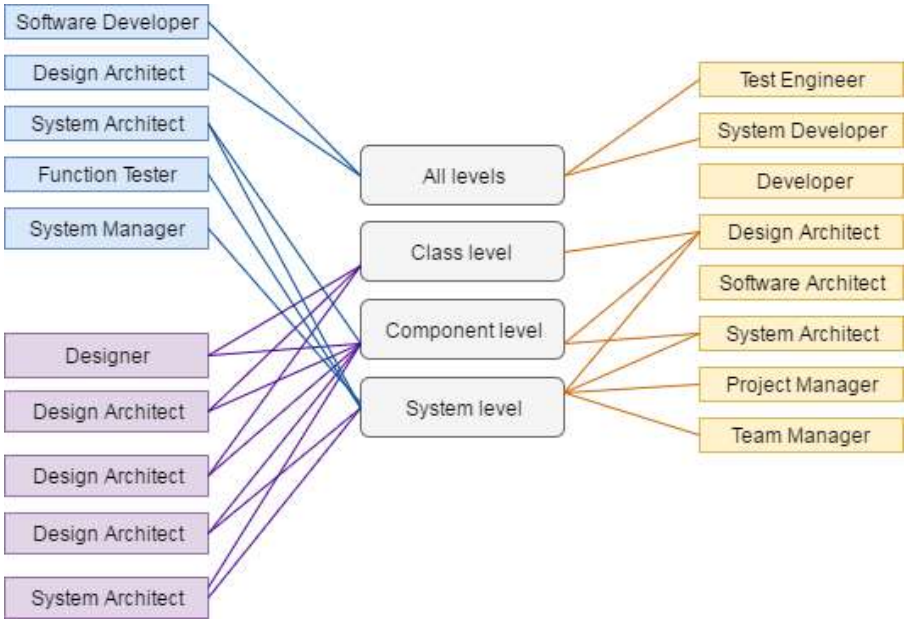

Figure 14. Level of detail required by a stakeholder for case 2-4.

Information need in Complementary category

Figure 13 displays additional information need that is not related to relationships between entities or structural composition of entities, or concerns changes-related information need. Based on the figure, test engineer requires the most complementary information, such as test coverage, most used parts of the code, implementation bottlenecks, and cyclamate number. Team manager requires visualization of implemented architecture in relation to requirements. 2 out of 3 software architects and 2 out of 5 design architects require view of "problematic" components, that have high maintenance cost or low test coverage, and impact of new flows on the old ones. One of the developers was interested Revision history and most CPU-heavy parts of the code, while another was interested in types of signals, ports and buses. Additionally, system designer was interested in types of signals as well and test engineer required information about revision history.

To summarize, for relationships category, relationships between components were the most frequently asked, following by systems and classes. On average, developers were interested in relationships between classes and packages; design architects – classes and components; system architects – systems, subsystems, and components; managers – systems and subsystems.
For composition category, component and system composition were mentioned most frequently, while class and composition to lesser extent. On average, developers were interested in composition of classes and packages; design architects – cluster of classes, classes, components; system architect – system, cluster of systems, components; manager – system.
For complementary category, developers are interested in types of signals, CPU-heavy parts of code; design architects- effect of a new flow on old flows, "problematic" components; system architects – types of signals, effect of new flow on old flows, "problematic" components, implementation in relation to requirements, and test coverage; managers – implementation in relation to requirements.

RQ3: What is the level of abstraction required from SAV depending on a stakeholder?

The data gathered during the interviews indicates level of detail required for each stakeholder, which is displayed in figure 14, however, level of detail will be determined in Discussion section, based on level of detail and information need. Case 1 did not provide information regarding their current visualization, and thus, results for this section are based on cases 2-4. According to the figure 14, both developers, one design architect and a test engineer require all levels of detail to support their work. Class level is required by a designer, design architect, and a developer, while component level was mostly requested by system and design architects. System level was most demanded, being used by a developer, design and system architects, function tester and management. Sorting by a stakeholder type, both developers required all levels of detail. Next, 3 out of 5 design architects required class and component levels, 2 out of 5 required system level, and one required all levels. All system architects required both component and system levels, and lastly, all managers required system level of detail.

To summarize, on average, developers required all levels of detail, design architects – class and components level; system architects, component and system levels; mangers – system level.

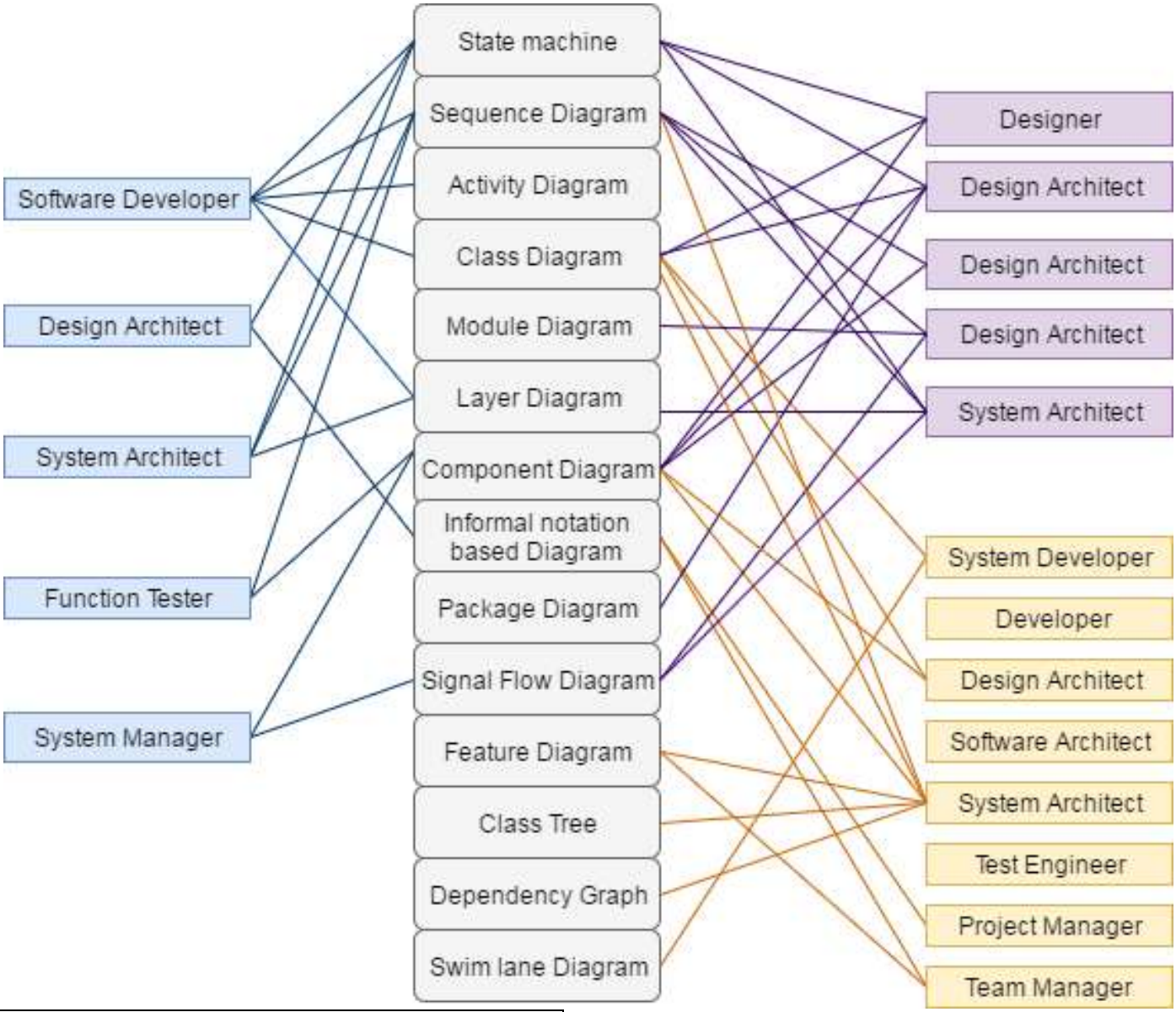

Figure 15. SAV techniques used in cases 2-4.

RQ4: What techniques of SAV can be employed depending on a stakeholder?

Figure 15 displays current SAV techniques employment by different stakeholders in Ericsson (case 2), Tetra Pak (case 3) and Ericsson (case 4). Interviewees from Volvo (case 1) did not provide any data regarding current SAV practices. Based on this figure, software developers most frequently employ state machine, sequence and class diagrams, with more rare instances of also employing activity and layer diagrams in case 4, and component diagram in case 2, and swim lane diagram in case 3. Design architects vary in techniques employed even more, with most frequent choice being Component diagram, being used by 3 design architects, state machine, class, and sequence diagrams, used by 2 design architects each. Less frequently used diagrams are package, module, and signal flow, with one design architect per diagram. All system architects in cases 2-4 used sequence diagrams and 2 out of 3 used state and layer diagrams. Layer diagrams were only used by system architects from Ericsson, due to layered architecture of their product. Signal flow, class, component, feature and dependency diagrams, and class tree were used by only one system architect. System manager used component and signal flow diagrams; team manager used feature diagrams. Both team and project managers used notation-based high-level abstract diagrams to communicate overall structure of a system. Test engineer and function tester used SAV techniques the least, with former using component and sequence diagrams and latter opting to not using SAV at all.

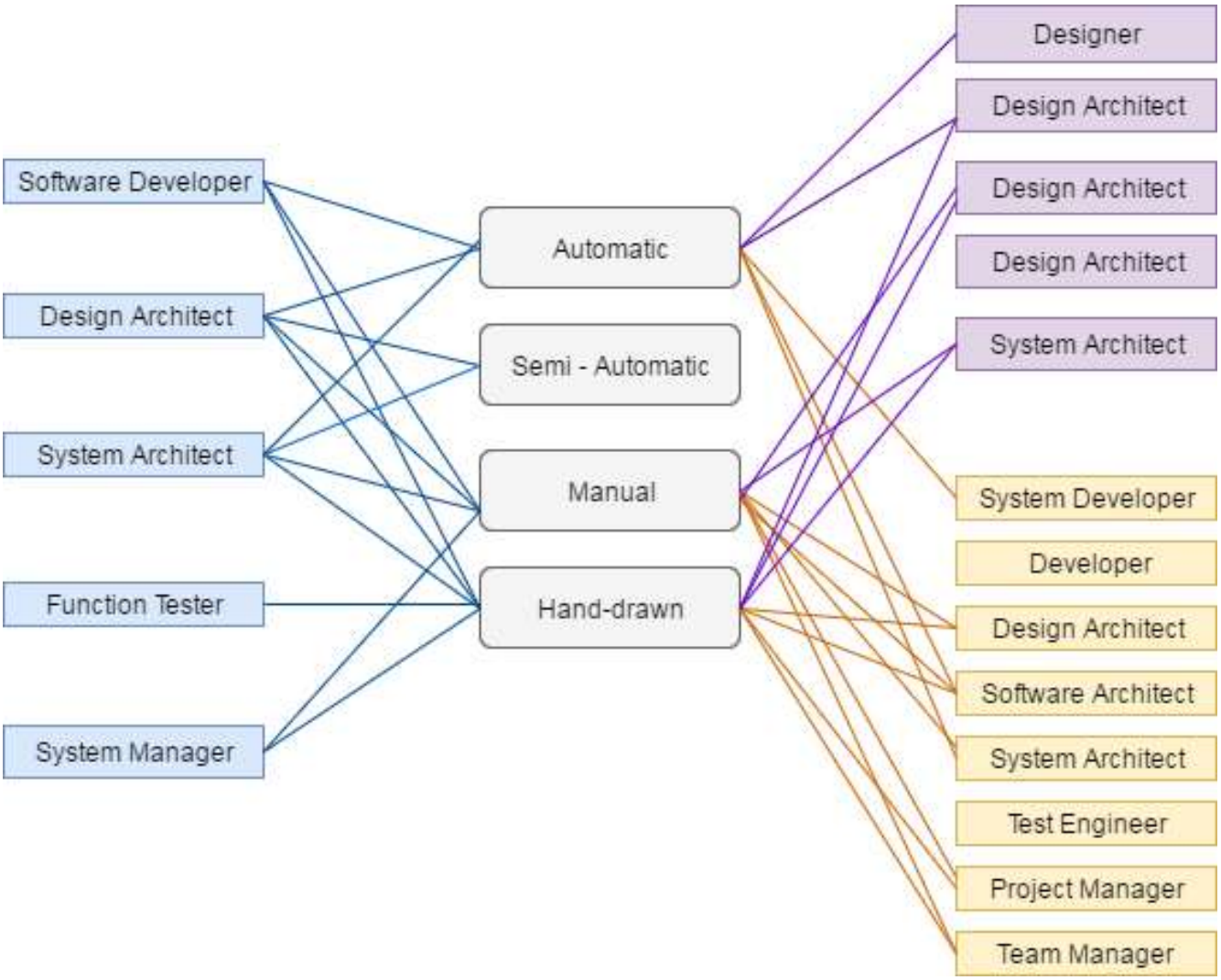

Figure 16. Types of SAV tools used by stakeholders in cases 2-4.

Component diagram was the most used diagram, with 9 stakeholders mentioning using it; sequence, state, and class are next most popular, with 6-7 users each; layer and signal flow are less frequently used, being mentioned by 3 stakeholders each. Least frequently used diagrams are activity, and module diagrams as well as class tree, dependency graph and swim lane diagram, being mentioned by one user each.

To summarize, components, state, sequence, and class diagrams are most frequently used diagrams. On average, developers used class diagrams the most; design architects – component, state machine, and sequence diagrams; system architects – state machine, sequence, and layer diagrams, managers – informal notation-based diagram.

All of the stakeholders employed notation-based techniques, majority of which was UML. System architect from case 3 was the only stakeholder to employ graph-based technique in addition to UML.

RQ5: What type of tools are used for SAV depending on a stakeholder? (Automatic, semi-automatic, manual)

According to Shahin et al. [6], automatic tools are capable of generating diagrams from the source code with minimal input from the user. Semi-automatic tools require user interference to a greater extent in order to provide additional configurations or alternative source for diagram generation, such as text-based description. Manual tools are entirely dependent on user input and usually are simple graphical tools, or tool features.

Similarly to previous section, case 1 dataset had no information on current SAV practices and or data on tools used, and therefore, only cases 2-4 are considered for this section. Additionally, case 2 design architect provided no data about tools he used; and case 3 developer and test engineer used no visualization, and thus, have no links to tool types.

According to figure 16, Automatic tools were used by 8 out of 15 stakeholders: by 2 out of 3 software developers, 2 out of 5 design architects, and 2 out of 3 system architects. In contrast, none of testers or managers used automatic tools. Semi-automatic tool was used exclusively by case 4 participants, specifically by design and system architect. Manual tools were used by 11 out of 15 stakeholders: 1 out of 3 developers, 3 out

of 5 design architects, by a software architect, by all system architects and managers. Lastly, hand-drawn diagrams were used by 12 out of 15 participants, being used by 1 out of 3 developers, 4 out of 5 design architect, and 2 out of 3 system architect, with addition of all the managers, software architect, and function tester.

However, these results do not necessarily reflect actual requirements of the stakeholders towards SAV tools, but only provides an illustration of current practices. In fact, many of the stakeholders noted that there is a gap in current tool support and suggested possible improvements or lacking tools. Most common suggestions and concerns were:

1. Low level diagrams are rarely maintained and get quickly outdated, becoming unreliable and incomplete. A solution for this can be, for example, to automatically generate diagrams from source code every time it is committed.
2. However, when generating low level diagrams, they are often very difficult to read.
3. Manually created diagrams are generally unreliable and should be avoided, which is also true for high level diagrams. Even though high level diagrams do not change as often as low level, they still should be generated.
4. There is a very small number of automatic tools available.
5. There is a need in a single tool that can substitute a collection of tools and be capable of automatic generation of different views and diagrams at a different levels of abstraction.
6. Elements in a diagram should be filtered in respect to stakeholder's concerns, displaying different areas of interest and metrics.
7. "Display-on-demand": hide unnecessary information, until it is requested.
8. Generate overlay diagrams: for example, sequence diagrams, with components diagram, which includes active components from sequence diagram.

To summarize, hand-drawn and manual tools were used the most, being mentioned by 12 and 11 stakeholders respectively; while automatic tools were used by 8, and semi-automatic by 2. However, majority of the stakeholders considered automatic tools support paramount.

On average, developers used automatic tools; design architects – hand-drawn, manual and automatic; system architects – manual, automatic, hand-drawn; managers – manual, hand-drawn. As a conclusion, developers were using more of automatic tools, while the rest of stakeholders employed more of manual and hand-drawn diagrams to a larger extent.

RQ6: What are the reasons for not employing SAV in the industry?

Participants from case 1 expressed their need in architecture visualization, but provided no information whether SAV was currently used to support their work. Therefore, the data for this research question was provided by cases 2-4. In these cases, 16 out of 18 interviewed stakeholders used software architecture to a various extent, with two exceptions being developer and test engineer from case 3. The developer had over 15 years of experience in development, was very familiar with the system and preferred reading code to visualizations. Test engineer, on the other hand, noted that he has a demand for visualization, but only if it provided additional information, such as test coverage and pointed at implementation flaws. Function tester from case 4 used visualization to a small extent, but stated that it was due to his personal interest in how the system's architecture is laid out. While performing functional testing, there was no need in knowledge of architecture, and thus, SAV was irrelevant to his direct responsibilities.

To summarize, 3 of the stakeholder did not use SAV to support their work. A developer did not see a need in it due to having a lot of development experience and preferring to read the code; test engineer did not use SAV because it was incapable of mapping metrics onto diagrams; and function tester did not use SAV because his responsibilities did not require knowledge of system's architecture.

VII. DISCUSSION

This section contains analysis of results described in the previous section, and its comparison with results of literature review. For most of research questions, the analysis will be carried out in a following manner: 1) Compare different stakeholders within same company; 2) Compare same stakeholders from different companies; 3) Compare this analysis with results of literature review. Firstly, this section includes discussion by research question and secondly, it includes discussion by stakeholder.

A. Discussion by Research Question

RQ1.What is the current demand for SAV in the industry depending on a stakeholder?

According to interview results, most of system architects found visualization useful when it comes to communication, understanding of architecture and decision making. Similarly, all managers found SAV useful in communication and decision-making, adding that it could be of even more value if it included mapped metrics. Design architects varied a little more in of how much value SAV is, stating that it is valuable as long as it kept up to date. Apart from one of the developers, who opted to not use SAV, other developers stated that it was

Table 1. Information need, level of detail, techniques, type of tools for stakeholders in cases 1, 2, sorted by cases.

		Information Need						
Case	Company	Stakeholders	Dependencies	Composition	Additional	Level of Detail	Techniques	Type of tools
Case 1	Volvo	System Designer	Packages, Systems	Package, Component, System	Types of Signals	-	-	-
	Volvo	Software Developer	Packages, Systems	Package, System	Types of Signals	-	-	-
	Volvo	Test Engineer	Systems	Component	Revision History, Implemetation in relation to requirements	-	-	-
Case 2	Ericsson	Designer	Classes, Components	Class, Component	-	Class level, Component level	State machine, Class, Component diagrams	Automatic
	Ericsson	Design Architect	Classes, Components	-	How new flow affects old flows	Class level, Component level	State machine, Class, Component, Package diagrams	Automatic, Hand-drawn
	Ericsson	Design Architect	Classes, Subsystems, Systems	Package, Cluster of Classes	-	Class level, Component level	Sequence, Component diagrams	Manua, hand-drawn
	Ericsson	Design Architect	Classes, Clusters of classes, Packages, Components	Classes, Cluster of Classes, Component	-	Component level, System level	Sequence, Module, Signal flow diagrams	-
	Ericsson	System Architect	Layers, Sybsystems, Systems	System, Cluster of Systems, Layers	How new flow affects old flows	Component level, System level	State machine, Sequence, Signal Flow, Layer diagrams	Manua, hand-drawn

Table 2. Information need, level of detail, techniques, type of tools for stakeholders in case 3.

		Information Need						
Case	Company	Stakeholders	Dependencies	Composition	Additional	Level of Detail	Techniques	Type of tools
Case 3	Tetra Pak	Developer	-	-	-	-	-	-
	Tetra Pak	System Developer	Classes, Packages	Class, Component, Assembly	Most CPU heavy parts of the code, Revision History	All levels	Class, Swim lane diagrams	Automatic
	Tetra Pak	Design Architect	Classes, Clusters of classes, Components	Class, Cluster of classes, Component, System	"Problematic" components	Class level, Component level, System level	Class, Component diagrams	Manua, hand-drawn
	Tetra Pak	Software Architect	Modules, Components, Subsustems	Component, System	"Problematic" components	-	-	Automatic, Manua, hand-drawn
	Tetra Pak	System Architect	Clusters of classes, Modules, Components, Systems	Package, Cluster of classes, Component, System, Cluster of Systems	Implementation in relation to requirements, Test Coverage	Component level, System level	Sequence, Class, Component, Feature, Class tree, Dependency graph	Automatic, Manual
	Tetra Pak	Test Engineer	-	-	Test coverage, Most used parts of the code, implementation bottlenecks, Cyclomatic number	All levels	-	-
	Tetra Pak	Project Manager	Sybsystems, Systems	System	-	System level	Informal abstract notation-based diagrams	Manua, hand-drawn
	Tetra Pak	Team Manager	Systems	System	Implementation in relation to requirements	System level	Informal abstract notation-based diagrams, feature diagram	Manua, hand-drawn

Table 3. Information need, level of detail, techniques, type of tools for stakeholders in case 4.

		Information Need						
Case	Company	Stakeholders	Dependencies	Composition	Additional	Level of Detail	Techniques	Type of tools
Case 4	Ericsson	Software Developer	Layers, Classes, Packages	Class, Package, Layers	-	All levels	State machine, Sequence, Activity, Class, Layer	Automatic, Manual, Hand-drawn
	Ericsson	Design Architect	Components	Class, Component, System	-	All levels	State Macine, Informal high-level notation-based diagrams	All
	Ericsson	System Architect	Layers, Modules, Components, Systems	System, Cluster of systems, Layer	-	Component level, System level	State Machine, Sequence, Layer diagrams	All
	Ericsson	Function Tester	Components	System	-	System level	Sequence, Component diagrams	Hand-drawn
	Ericsson	System Manager	Components, Subsystems	System	-	System level	Component, Signal flow diagrams	Manua, hand-drawn

Table 4. Information need, level of detail, techniques, type of tools for developers.

		Information Need						
Case	Company	Stakeholders	Dependencies	Composition	Additional	Level of Detail	Techniques	Type of tools
1	Volvo	Software Developer	Packages, Systems	Package, System	Types of Signals	-	-	-
2	Tetra Pak	Developer	-	-	-	-	-	-
	Tetra Pak	System Developer	Classes, Packages	Class, Component, Assembly	Most CPU heavy parts of the code	All levels	Class, Swim lane diagrams	Automatic
4	Ericsson	Software Developer	Layers, Classes, Packages	Class, Package, Layers	-	All levels	State machine, Sequence, Activity, Class, Layer	Automatic, Manual, Hand-drawn

Table 5. Information need, level of detail, techniques, type of tools for design architects.

		Information Need						
Case	Company	Stakeholders	Dependencies	Composition	Additional	Level of Detail	Techniques	Type of tools
2	Ericsson	Design Architect	Classes, Components	-	How new flow affects old flows	Class level, Component level	State machine, Class, Component, Package diagrams	Automatic, Hand-drawn
	Ericsson	Design Architect	Classes, Subsystems, Systems	Package, Cluster of Classes	-	Class level, Component level	Sequence, Component diagrams	Manua, hand-drawn
	Ericsson	Design Architect	Classes, Clusters of classes, Packages, Components	Classes, Cluster of Classes, Component	-	Component level, System level	Sequence, Module, Signal flow diagrams	-
	Ericsson	Designer	Classes, Components	Class, Component	-	Class level, Component level	State machine, Class, Component diagrams	Automatic
3	Tetra Pak	Design Architect	Classes, Clusters of classes, Components	Class, Cluster of classes, Component, System	"Problematic" components	Class level, Component level, System level	Class, Component diagrams	Manua, hand-drawn
4	Ericsson	Design Architect	Components	Class, Component, System	-	All levels	State Macine, Informal high-level notation-based diagrams	All

Table 6. Information need, level of detail, techniques, type of tools for system architects.

		Information Need						
Case	Company	Stakeholders	Dependencies	Composition	Additional	Level of Detail	Techniques	Type of tools
1	Volvo	System Designer	Packages, Systems	Package, Component, System	Types of Signals	-	-	-
2	Ericsson	System Architect	Layers, Sybsystems, Systems	System, Cluster of Systems, Layers	How new flow affects old flows	Component level, System level	State machine, Sequence, Signal Flow, Layer diagrams	Manua, hand-drawn
3	Tetra Pak	Software Architect	Modules, Components, Subsustems	Component, System	"Problematic" components	-	-	Automatic, Manua, hand-drawn
	Tetra Pak	System Architect	Clusters of classes, Modules, Components, Systems	Package, Cluster of classes, Component, System, Cluster of Systems	Implementation in relation to requirements, Test Coverage	Component level, System level	Sequence, Class, Component, Feature, Class tree, Dependency graph	Automatic, Manual
4	Ericsson	System Architect	Layers, Modules, Components, Systems	System, Cluster of systems, Layer	-	Component level, System level	State Machine, Sequence, Layer diagrams	All

Table 7. Information need, level of detail, techniques, type of tools for managers.

		Information Need						
Case	Company	Stakeholders	Dependencies	Composition	Additional	Level of Detail	Techniques	Type of tools
3	Tetra Pak	Project Manager	Sybsystems, Systems	System	-	System level	Informal abstract notation-based diagrams	Manua, hand-drawn
	Tetra Pak	Team Manager	Systems	Team Manager	Implementation in relation to requirements	System level	Informal abstract notation-based diagrams, feature diagram	Manua, hand-drawn
4	Ericsson	System Manager	Components, Subsystems	System	-	System level	Component, Signal flow diagrams	Manua, hand-drawn

Table 8. Most common information need, level of detail, techniques, type of tools for each stakeholder.

Information Need						
Stakeholders	Dependencies	Composition	Additional	Level of Detail	Techniques	Type of tools
Developer	Classes, Packages	Classes, Packages	Types of Signals, Most CPU heavy parts of the code, Revision History	All levels	Class diagram	Automatic
Design Architect	Classes, Components	Cluster of classes, classes, components	How new flow affects old flows, "Problematic" components	Class level, Component Level	Component, State Machine, Sequence diagrams	Hand-drawn, manual, automatic
System architect	Systems, Subsystems, Components	System, Cluster of systems, components	Types of Signals, How new flow affects old flows, "Problematic" components, implementation in relation to requirements, test coverage	Component, system level	State machine, sequence, layer diagrams.	Manual, automatic, hand-drawn
Manager	Systems, Subsystems	System	Implementation in relation to requirements	System level	Informal abstract notation-based diagrams	Manua, hand-drawn

definitely useful to support their work mainly by aiding understanding of architecture and communication.

To compare same stakeholders from different companies, there was little difference for system architects, despite little SAV employment in case 3. Similarly, system managers have similar attitude towards SAV across different cases. Design architects vary in their attitude even within same company, however, the design architect that had lower demand in SAV also shared developer's responsibilities, which could affect his view. Lastly, developer from case 4 stated that visualization was definitely useful for him, while developers from case 3 stated that it was useful to some extent.

As a results, it was observed, that higher level stakeholders, such as managers and system architects had higher demand for SAV regardless the company; while developers found SAV less useful in a case with less architectural guidance. Design architects varied across different cases and within same cases due to having additional individual requirements due to their personal responsibilities and areas of interest.

To compare with reviewed literature, developers are primary

users of SAV, followed by testers, software architects and managers according to Mattila et al. [2], however, this is not observable in the data. Lowered interest in SAV from developers can be explained by lack of appropriate tools and techniques, which cater to their needs. This view is shared by Merino et al. [33], who adds, that "efforts in software visualization are out of touch with the needs of developers", which is discussed more in detail in later subsections, dealing with tools support.

RQ2.What is the information need of different stakeholders towards SAV?

Developers

As it was mentioned before, developers are interested in visualization of code changes and their impact and aiding system comprehension.

Based on the data in the tables 1-7, information needs of developers are homogeneous to a great extent across different companies when it comes to "Relationships" and "Composition" categories. Former category included only a small variety of needs, such as relationships between classes, packages, systems and layers, with classes and packages being mentioned most frequently. "Composition" needs were of slightly greater variety, including Class, package, component, assembly, system and layer, with classes and packages being most frequently mentioned as well. The differences in information need, particularly in "complimentary" category of information needs, could be attributed to product's domain, or developers' specific responsibilities. For example, a developer from case 1 is interested in tracking signals between different ECUs and viewing information about different types of signals, which does not apply to case 3 system developer, who is more concerned with performance and would like to visualize parts of the code, which are most CPU-heavy. Therefore, while developers' needs in "Relationships" and "Composition" categories are quite homogeneous with a visible pattern of interest in composition of and relationships between classes and packages, it is more difficult to find common ground in "Complementary" category.

According to Gallagher et al. [7], developers are most interested in static and dynamic visualizations at the code level, which is confirmed by interview data. However, LaToza and Myers [43] claim that developers require visualization in "Change" category, with lesser interest in "Composition" and "Relationships" categories, signifying that the some of the most relevant questions for developers are:

1. Why was this done this way?
2. When, how and by whom was this code changed or inserted?
3. How has it changed over time?
4. Have changes in another branch been integrated into this branch?
5. What are implications of this change?
6. Is the existing design a good design?
7. Is this tested?

Considering LaToza and Myers point, there can be a significant gap in the gathered data, that might indicate very small sample size, which is not reflective of real information need of developers; or this might be attributed to the method of gathering information, since interviews might not allow time for an interviewee to reflect.

Architects

According to reviewed studies, architects require higher level of visualization, displaying attributes of a designed architecture, such as complexity, coupling and cohesion [8]. Appropriate visualization can also aid identifying components for reuse [21], software architecture documentation [6, 22, 8], and monitoring software architecture evolution [6, 22, 2]. Overall, architects require visualizations that enable navigation of "software structure, dependencies, and attributes such as quality metrics [5]".

According to the data in the tables 1-7, design architects have relatively similar information needs in terms of relationships between entities, including relationships between classes, clusters of classes, packages, and components, with classes and components being mentioned the most frequent. Similarly, for "Composition" category, the most frequently mentioned entities were classes, clusters of classes, and components, while composition of systems was less frequently mentioned. From the gathered data, it is visible that design architects are somewhat a heterogeneous group in terms of their need towards "Relationships" and "Composition". Even though there is common need for most of these stakeholders, some requirements still vary, which could be attributed to difference in domains, and task distributions. Some of the interviewees that also took on responsibilities of developers had requirements similar to those of developers, as well as selecting automatic tools. For "Complementary" category, only "problematic components" and "how new flow affects the old flows" were mentioned. These requirements are similar to LaToza and Myers [43] questions in "Change" category in particularly question number 5. "What are implications of this change?"

System architects were interested in looking at software entities at a higher level, most frequently requiring visualization of relationships between systems, subsystems, and components, with lesser interest in layers (specific for cases 2 and 4), and modules. For "Composition" category, the primary interest was system, cluster of systems and component, with lesser interest in packages, clusters of classes, and layers. Variety in interest for this stakeholder could be attributed to a number of things: 1) personal interest and experience, as it was for one of the interviewee with an interest in acquiring lower level understanding of the system; 2) task distribution and role separation issues: one of the interviewees held title of system architect, but was mostly involved in development, which explained his interest in lower level visualizations. In general, across companies, this stakeholder type was quite homogeneous in terms of information need in "Relationships" category, and semi-

homogeneous in "Composition" category. In contrast to previous stakeholders, system architects named more requirements in "Complementary" category. This can be explained by interest in quality attributes and metrics [8, 5] in case of "Problematic components" visualization requirements; identifying components for reuse [21] in case of test coverage requirement; navigation of dependencies in case of "how the new flow affects the old flows", and "implementation in relation to requirements" in case of documentation and communication.

Managers

According to Ghanam and Carpendale [8]: managers monitor progress and determine whether goals were completed; view problematic components, that can have high costs associated with development and maintenance; and understand time estimates for implementation [6]. Overall, in case of project managers, SAV should support monitoring of evolution of a system over extended period of time, providing information about general trends, such as "architectural erosion, rule violation, and quality decay" [5].

Based on the results in the tables 1-7, most common information need in "Relationship" category is relationships between systems, and subsystems, with lesser interest in components. For "Composition" category, all of interviewees required visualization of system composition. Surprisingly, only one interviewee mentioned need in third category, which was not expected due to manager's need in visualization of metrics, such as cost and performance. Overall, this stakeholder type was the most homogeneous in terms of information need than others, which could be explained by their distance from design and implementation, which is more divisive due to differences in domain and practices.

Testers

Testers is most heterogeneous group, due to different stages of testing they perform. Case 4 tester is performing function test with no need of support of SAV, however, he still uses abstract high-level visualization to improve understanding of the system. Case 1 test engineer is requiring visualization of dependencies between systems and composition of components. Case 3 test engineer did not employ any visualization at the moment of the interview, yet listing his needs in "Complementary" category. This category seemed to be particularly important for testers due to need in visualizing metrics, such as test coverage and cyclomatic complexity, as well as revision history, implementation bottlenecks, most used parts of the code, and implementation in relation to requirements.

RQ3.What techniques of SAV can be employed depending on a stakeholder?

Developers

In terms of preferred techniques, developers are a heterogeneous group to some extent, using a variety of diagrams such as class, swim lane, state machine, sequence, activity and layer diagrams, with class diagram being the only explicit common ground. Swim lane, depending on how it is used, can be similar to sequence, activity, and state diagram in terms of information they provide, and thus, can be considered another common technique. Both developers are concerned with tracking dependencies, but in contract, case 4 developer has a personal interest at viewing "how low level fits into the system" and carry out maintenance tasks, which can explain additional need in layer diagrams. Another explanation of the difference could be general difference in practices between two cases, since all stakeholders in case 3 employed less SAV than case 2 and 4, and thus can be more prone to choosing relatively informal swim lane diagram to more formal UML alternatives. Another possible reason for differences in choices of techniques, or number of techniques could be experience. One of the developers in case 3, that did not use any visualization linked it to experience and good understanding of the system, which allowed him to build his knowledge based on reading code only. Even though all of the interviewed developers were experienced, relatively less experienced developers employed more visualization. This could be another link to explore further in, however, current data sample is not substantial enough to arrive to any concrete conclusions in this regard.

Despite graph-based techniques being the central focus of research community in SV field and having highest automatic tool support [6], majority of stakeholder used UML to support their work. From developers' perspective, UML is sufficient for low-level visualizations, especially in case of inter-class relationships and composition, for which it was first developed [35, 38]. However, when generating diagrams from high number of entities, resulting diagrams can be difficult to read, due to "the amount of textual information depicted by each component" [38]. This opinion was repeated by both developers, stating that generated class diagrams should be sorted or condensed.

Another solution a cluttered generated class diagrams are treemap, clustered graph and hierarchically bundled edges techniques, that are capable of automatically generating readable visualizations from 1000 software entities [5], which is paramount to developers' interests, according to Telea et al.[5]. Bundled edges and clustered graph techniques provide information about composition and relationships between entities, but are also complemented with metrics and attributes, while offering intuitive navigation. Metrics and attributes are confirmed to be of interest to developers by the interview data and by Telea et al. [5], stating that "views of code, metrics, structure, and dependencies" are indispensable to this stakeholder group. Besides bundled edges and clustered graph layout, there are techniques that combine familiar UML diagrams with metrics or areas of interest, that are presented with supporting tools by Byelas and Telea [20] and Termeer et al. [42]. Another type of technique that can support developers' interest is metaphor based 3D visualizations that

offer a view on composition, relationships and metrics, however, it is unclear if 3D techniques provide better understanding and more intuitive navigation than other techniques [24].

Architects

Design architects are heterogeneous in their employment of SAV techniques, including class, package, components, state machine, sequence, module, signal flow, and high level notation based diagrams, with component, state machine, and sequence diagrams being most frequently mentioned. Considering differences between design architects from different companies, there are no obvious patterns, except that only case 2 design architects used sequence diagrams, and generally had more variety in techniques. This can be attributed to an established practices of documentation and communication within a company. Another factor could be domain or complexity of products, but there is no concrete data available to compare complexity of products developed by cases 2 and 3. Secondly, design architects that were also involved in development, such as in case 3, preferred lower level visualizations, such as class diagrams.

System architects were even more heterogeneous in terms of techniques they preferred, including state machines, sequence, signal flow layer, class, component, and feature diagrams, with most frequently mentioned techniques being state machine, sequence and layer diagrams. System architects from cases 2 and 4 had the most similar demands, which can be attributed that both cases belonged to the same company. Case 3 system architect, however, used bigger variety of techniques, with some of them being at a class level, which could be attributed to task distribution in case 3, in which many interviewees were responsible for multiple stakeholders' tasks. In this example, case 3 system architect dealt more with development, than architecture, and thus had "a developer's perspective". With this in mind, if case 3 architect is not considered, system architects use similar diagrams, such as state machine, sequence, and layer (applies for case 2 and 4 only) diagrams.

All design and system architects, with an exception for one, used UML or informal notation-based diagrams. Although none of the interviewees expressed dissatisfaction with UML visualization explicitly, some of reviewed papers [35, 38] argued that UML offers insufficient support to higher level diagrams. Based on the gathered data, higher level diagrams are hand-drawn component diagrams, or abstract diagrams using package or class notations, as "same notation can have a wide range of semantics" [44]. However, close to a half of stakeholders stressed out the importance of generation of diagrams, even at a high level of abstraction, which is difficult to do employing UML. Furthermore, UML is deficient to display "general sequence of activities and dynamic aspects of the structure" [44], which are relevant to architects' interests.

Another need of this stakeholder group was visualization of metrics, based on interview data and reviewed research [5, 8], however, none of used techniques supported this need. According to Carpendale and Ghanam [8], architects require higher level of visualization than developers, complemented by display of attributes, such as complexity, coupling and cohesion [8]. Previously mentioned hierarchically bundled edges and clustered graph techniques can satisfy needs of both developers and architects, due to ability of "zooming in and out", producing diagrams at different level of detail. An advantage of using same tool/techniques by different stakeholders is providing a common set of diagrams that different stakeholders can navigate easily, that improves communication. Similar layout across different diagrams does not distort user's "mind map" and contributes to a more complete understanding of a system from different points of view [21].

Another technique, supporting needs of architects was presented with accompanying tool by Lungu and Lanza [41]. This technique allows viewing clusters of classes or modules with similar behaviors and relationships between them, as well as visualizing "the evolution of an inter-module relation through multiple versions of the system".

Managers

Managers were the most homogeneous group, in terms of information need, but had more variety in techniques they used. The techniques included component, signal flow and feature diagrams, but the most frequently mentioned were informal abstract notation-based diagrams that conveyed general structure of a system.

Since only case 3 and 4 provided data regarding needs of this stakeholder group, it is difficult to generalize and compare between different companies. Both case 3 managers used abstract notation-based diagrams, while case 4 manager used formal UML diagrams, which could be attributed to differences communication and documentation standards and processes between the two companies.

In comparison to other stakeholders, managers use more abstract, informal diagrams that include visualizations of relevant components to a specific requirement, due to responsibility of in monitoring "progress of the project and determine the completion of the development goals"[8]. However, none of the managers expressed a need in visualization of architecture in conjunction with metrics, which contradicts a statement, that managers require visualization of cost-heavy components and some other quality metrics [5, 21]. Additionally, none of used techniques were supporting efficient visualization of architecture evolution, which was stated as one of manager's' concerns [5, 8]. To address this need, previously mentioned Langu and Lanza's [41] filmstrip displays "the evolution of an inter-module relation through multiple versions of the system". Additionally, according to Telea et al.[5], managers are interested in methods of visualization that display abstract composition of a system, or distribution of artifacts, with

addition of relevant attributes and metrics, such as treemaps and dense pixel charts, which are capable of displaying big set of data [25].

Testers

Since only 1 out of 3 testers provided data about current employment of SAV techniques, it is impossible to compare and contrast different cases. Case 4 tester used sequence and component diagrams, however, it was not necessarily to support his work, as he dealt with function testing, but to enhance his understanding of the system due to personal interest.

RQ4.What is the level of abstraction required from SAV depending on a stakeholder?

As it was mentioned earlier, in this paper, 3 levels of abstraction are defined based on Gallagher et al. [7], which are:

1. Low level of abstraction, which is source code level visualization, that related directly to a software artifact;
2. Middle level, which is problem-specific visualization of a particular area of interest, such as a sequence diagram of a particular flow;
3. High level, which communicates design decisions and metrics in addition to overall structure of a system.

Gathered interview data included level of detail, information need, and techniques preferred by different stakeholder. These data points in conjunction determine level of abstraction for each stakeholder type.

Developers

Both developers, that provided data about current SAV employment, required visualizations at all levels of detail, while using class and sequence diagrams to a large extent. Their information need in composition and relationships was mostly related to classes and packages, with less frequent need in metrics, CPU-heavy parts of the code. Based on this data, developers require both low and medium levels of abstraction. Need in high level of abstraction for developers might vary greatly due to their background and interests. One of the developers was interested in seeing "how low level fits into the system", while another was interested in metrics, provided by high level visualization.

Architects

Design architects required mostly class and component level of detail, with system level to a slightly lesser extent. Component level was the most frequently mentioned, with additional alternating between class or systems levels by 3 out of 5 design architects; while 2 other design architects required all levels of detail. Component, state machine, and sequence diagrams were used the most, with greater need for metrics display, than developers. Based on this data, design architects require all 3 levels of abstraction, which was also stated explicitly by 2 design architects during the interviews.

Need for lower level of abstraction and detail can be explained by specific tasks performed by each design architects, as they can assume responsibilities of development, or architectural design closer to the system level. This pattern is traceable in the data, as design architects that had development and testing responsibilities preferred lower level of detail.

System architects were very homogeneous in terms of level of detail required, choosing component and system levels. Preferred techniques most commonly included state machine, sequence and layer diagrams, in addition to highest demand for metrics display. Therefore, system architects prefer high and medium levels of abstraction. However, one of system architects also required low level visualizations that could be explained by his responsibilities of development in parallel to architecture.

Managers

As it was mentioned before, this group was most homogeneous with all requirements, and no outliers. Their information need was concentrated on composition and relationships between systems with mapped metrics, at a system level of detail, communicated via informal notation-based techniques. Based on this data, managers require high level of abstraction.

RQ5.What type of tools are used for SAV depending on a stakeholder? (automatic, semi-automatic, manual)

Developers

Interviewed developers used automatic, manual, and hand-drawn diagrams, with automatic being the most frequent. Many of the stakeholders, developers in particular, stressed the importance of automatic tools. Whether hand-drawn diagrams seem to be vital for most stakeholders to aid communication, manual tools fill the gap where automatic tools lack, and kept for purposes of documentation.

According to Telea et al. [5], developers are interested in viewing automatically generated uncluttered diagrams of "correlated structure, dependency, metrics", by a tool that requires minimal user intervention, and are IDE integrated. Although developers' requirements for tools are most satisfied in comparison to other stakeholders [5,7], "developers have little support for adopting a proper visualization for their needs", currently presented tools are "out of touch with the needs of developers" or developers are simply "unaware of existing visualization techniques to adopt for their particular needs" [33]. It is possible, that commercial tools tend to support well-established techniques of visualization, such as UML, which is not necessarily suiting developers', or higher level stakeholders' requirements. At the same time, majority of innovative tools, that are not limited to UML and suitable for supporting developers' needs are developed as a part of

research community and are maintained for a limited period of time, which might prevent companies from investing time into these tools.

Architects

Design architects reported using all tool types, with hand-drawn diagrams being mentioned most frequently, and manual and automatic to a lesser extent. While using hand-drawn and manual tools to similar extent as automatic, most design architects considered manual methods of diagrams creation a waste of time, as well as unreliable, and incomplete. This stakeholder type emphasized the importance of automatic generation of diagrams and continuous updating of diagrams for them to be relevant. Another requirement was to supply a single tool that would be able to generate different views automatically, as well as show dependencies at different levels of abstraction. Lastly, it was observed, that there are low-level and high-level views available, but nothing of middle-level, problem-specific.

System architects used manual, automatic and hand-drawn diagrams to same extent. Similarly to design architects and developers, system architects emphasized the importance of automatic tool support and continuous diagram updating. From their perspective, manually created diagrams are unreliable, incomplete, take time, and get outdated very quickly. The currently available automatic tool were few in number and provided little support for generation of high-level overview of a system. Another requirement was to visualize metrics, such as complexity on lower level and dependency count of higher level diagrams, among others. Lastly, it was important to both design and system architects to view automatically generated problem-specific diagrams, or enable filtering and searching diagrams.

According to Telea et al. [5], lead architects require automatic tools, that assist in discovering "evolution problems" and display metrics as well as similar to developers' tools, that allow "high visual scalability". Tools that would support techniques appropriate for this stakeholder type, such as hierarchically bundled edges, and clustered layout graph, that are highly scalable and enable visualizations at different levels of abstraction and displaying metrics. Additionally, Metric View and a tool developed by Byelas and Telea [20] are UML-based tools that group entities by metrics and areas of interest, capable of displaying architecture at different level of detail.

Managers

All interviewed managers used manual and hand-drawn diagrams. Although managers require high-level overview of a system, which changes infrequently, and none mentioned an interest in metrics, a need for automatically generated visualization was still pointed out.

Another requirement, similarly to architects, was to support middle-level of abstraction that would allow visualization of problem-specific information.

According to Telea et al. [5], managers require a tool that visualizes "multivariate plots of processes and product" with minimal user input, performs automatic "fact extraction from repositories" and maps it onto abstract architectural overview.

RQ6.What are the reasons for not employing SAV in the industry?

Based on the data in tables 1-7, 3 out of 18 stakeholder, that provided information about current SAV practices (cases 2-4), did not use visualizations to support their work.

Two of these stakeholders were testers. One of the tester from case 3 did not employ visualizations because currently used tools did not support mapping metrics to architecture diagrams, which were of particular interest to this tester. However, she stated that if that was to be supported, architecture diagrams with mapped metrics could be of great value. Another tester from case 4 did not employ SAV to support his work, because his responsibilities of function testing required no knowledge of architecture.

Another stakeholder who did not use SAV to support his work was a developer in case 3. He attributed his lack of need in SAV to being experienced in development and having a good understanding of the system he was working with, which allowed him to base his knowledge on reading code only. However, another developer, with greater experience both in development and same experience with the system did use SAV to support his work. Thus, based on this data, it is not completely warranted to claim that there is a strong correlation between level of experience and to what extent SAV is used by a developer.

B. Discussion by Stakeholder type

Developers' requirements towards software architecture visualization

When studying requirements of developers, it was observed that their information need in relationships and composition categories concerned classes and packages. Additional information need that was currently lacking or not supported by employed techniques included types of signals, CPU heavy parts of the code, and revision history. All developers used UML, with class diagram being most preferred. This stakeholder type required all levels of detail and low and medium levels of abstraction. High level of abstraction was mentioned as well, but varied from case to case and based on personal interests. Automatic tool were used to a slightly greater extent than hand-drawn diagrams, however, it was most likely to lack of available IDE-integrated automatic tools that are able to generate readable diagrams from large number of entities quickly, and not to an actual preference to hand-drawn and manual tools. Requirement of wider selection of

automatic tools was stressed by all developers, which used SAV to support their work.

As a group, developers had somewhat varying concerns from one another, which could be explained by domain or specifics of the products they are working, for example, a developer from case 4 was interested in viewing layer diagram, due to layered architecture of the product; and a developer from case 1 was interested in viewing types of signals due to being involved in work related to embedded systems. Furthermore, it is important to distinguish between used and preferred diagrams/techniques, since it is not clear whether choice of diagrams was influenced by workplace practices or standards.

Based on these requirements, additionally to currently employed methods of visualization, graph-based hierarchical edge bundles and clustered graph layout can be used to effectively communicate composition and relationships between large numbers of entities at a different levels of abstraction.

Design Architects' requirements towards software architecture visualization

It was observed, that design architects require information about composition of clusters of classes, classes and components most frequently and relationships between classes and components. Additionally they are interested in visualizing implications of new flows to old flows, and "problematic" components. This stakeholder group was mostly interested in class and component level of detail, while requiring all levels of abstraction, which was stated explicitly by two design architects. All of these stakeholders used UML, mostly preferring component, state machine, and sequence diagrams. Majority of design architects used hand-drawn diagrams, then manual and automatic. However, similarly to developers, this was most likely not done due to actual preference for hand-drawn and manual tools, but lack of tools that appropriately support their needs. In terms of tools, design architects required automatic tools that would substitute a collection of tools, and generate diagrams at different levels of abstraction.

As a group, design architects are quite similar in their information needs and needed level of detail/abstraction, only sometimes requiring information about composition and relationships between systems. Automatic tools were used by all design architects except for 2 instances where design architect was concerned with higher level visualizations or belonged to case 3, which generally had very little automatic tool employment.

Based on these needs, design architects could also employ graph-based hierarchical edge bundles and clustered graph layout techniques that can automatically generate diagrams at different levels of abstraction with mapped metrics. Additionally, more familiar UML-based technique/tool MetricView is capable of automatic visualization of architecture in conjunction with selected metrics. Semantic dependency matrix can also be a useful technique/tool for automatically visualizing dependencies at different levels of abstraction.

System Architects' requirements towards software architecture visualization

When studying needs of system architects, it was observed, that their information need encompassed composition of systems, clusters of systems and components and relationships between systems, subsystem, and components. This stakeholder type had the highest demand for additional information and metrics, including types of signals, implications on a new flow to the old ones, "problematic" components, implementation in relation to requirements and test coverage. They required component and system level of detail and medium and high level of abstraction. One of system architects also required low level of abstraction, which could be explained by his additional responsibilities as a developer. This stakeholder level used a widest array of techniques, most of which were UML diagrams such as state machine, sequence, and layer diagrams and few graph-based diagrams. Although, UML is still dominating techniques used for SAV, use of graph techniques by system developers can be explained by UML insufficiency when it comes to visualization of high level architecture overviews. Manual tools were used the most, with automatic and hand-drawn to a lesser extent. However, it was explicitly stated by almost all system architects that it is very important to have diagrams automatically generated and that there are not enough available automatic tools.

Other requirements were to show "details-on-demand" as a method of compression and improving readability of generated diagrams; and filtering and searching automatically generated diagrams.

Considering these requirements, system architects require tools that are able to automatically generate and display scalable diagrams at different levels of abstraction, with mapping of high number of metrics. Similarly to design architects and developers, graph-based hierarchical edge bundles and clustered graph layout techniques can be used for these purposes, as well as improve communication between different stakeholder due to basing communication on the same layout and a common set of diagrams. Additionally semantic dependency matrix can be used to track dependencies between different software entities at different level of abstraction. Lastly, edge evolution filmstrip can be used to monitor dependencies across different versions of a system.

Managers' requirements towards software architecture visualization

When studying managers' requirements towards SAV, it was observed, that composition of systems and relationships between systems and subsystems were the most frequently mentioned. Additionally, visualization of implementation in relation to requirements was requested. System level of detail and high level of abstraction was preferred. All managers used manual and hand-drawn tools, but at the same time emphasized the need for automatic tools and continuous automatic update of diagrams.

As a group of stakeholders, they had the most similar interests and request, with minimal variation. Only substantial difference was that case 3 managers used informal notation-based visualizations, while case 4 managers used UML, which could be attributed to general lack of visualization practices in case 3.

Based on this data, managers require automatically generated visualization which is capable of integrating multiple quality-related metrics and tracking systems' evolution. For this stakeholder, clustered graph layout is capable of visualizing large systems in conjunction with multiple metrics at a high level of abstraction.

VII. CONCLUSION

This paper examined requirements of stakeholder towards software architecture visualization tool and techniques as well as information need and required level of abstraction. For this purpose, 21 interviews with stakeholders such as developers, design architects, system architects, managers, and testers were analyzed and compared to each other as well as to results of literature review of related studies. As a result, this paper contributes with knowledge of different stakeholders' information need and required level of abstraction, which was previously lacking in the current body of knowledge; and provides practical implications that might be of use to tool vendors, or practitioners that are looking to employ software architecture visualization to support their work.

A. Summary of findings regarding stakeholders' needs

There was an observable difference between stakeholders' requirements in a same company due to separation of concerns. However, there also were differences between same stakeholders across different companies, possible reasons for which are discussed below.

Overall, developers shared similar information needs, but employed different techniques to satisfy them, generally preferring automatic tools; design architects shared relatively similar information needs in "composition" and "relationship" categories, techniques and level of detail, but different needs in "complimentary" category and used tools; system architects have similar information needs and very similar level of abstraction requirements, but use widely different techniques; managers had the most similar requirements and practices, with lesser similarity in techniques.

B. General observations

Majority of differences between stakeholders across different cases could be attributed to the following:

1. Practices and standards: set practices to follow within an organization can influence techniques or tools that are used by stakeholders. For example, if it was customary to use a specific diagram/technique for documentation or communication, it is likely, that a stakeholder would conform to these practices.

2. Position title vs. Responsibilities: different companies can distribute responsibilities in a different way. For example, in case 3 system architect was mostly responsible for development, rather than architecture design, while some interviewees from the same case that were responsible for development, also took part in architectural design. Similarly, design architect from case 2 had development responsibilities.

3. Personal interests: some stakeholders may have an interests in parts of a system or a process which is not directly linked to their responsibilities, which affects their information needs. For example, a manager in case 3 was more interested in architecture than managers from other cases. A function tester, which did not necessarily required knowledge of architecture, still used architecture visualization for personal interest.

4. Domain-specific information: stakeholders in from different cases may have additional information needs, such as CAN frames for case 1.

5. Complexity of a system: complex systems can require more rigorous documentation and more visual aid for communication, which prompts stakeholders to use SAV to a greater extent.

Based on interview results and literature review results, the main issues when adopting visualization is automatic tool support. All stakeholder groups emphasized the importance of automatic tools support, however, only developers use automatic tools to a greater extent than other types of tools. Many of automatic, innovative tools, which cater to various needs of stakeholder, and might be more efficient than currently employed tools/techniques are developed as part of research community, but maintained for a short periods of time.

C. Threats to Validity

It is acknowledged that case study has a number of disadvantages, such as bias and difficulties when it comes to generalization [15]. It is also required to address threats to validity. From internal validity the following factors may undermine validity:

1. History: factors outside of the study, such as personal experience, company's standards and procedures can affect how stakeholders view and use SAV, An effect of this can be that same stakeholder can have different information needs, or preferred techniques. However, such differences were accounted for and analyzed, and thus, pose little threat to validity.

2. Interviewers change: the interviews were carried out by different interviewers which could affect pace or structure of the interview and thus produce inconsistency in the results. A strategy to mitigate this was to follow common interview guide as previous interviews, with exceptions for a few additional questions.

3. Selection bias: interviewees for case 4 were selected by contact person in the company, and not the authors of this paper. This eliminated bias from researchers' side, but it is unclear whether there were unknown criteria of selection on the company's side besides stakeholder title. To some extent this can be mitigated by considering interviewees' background, such as experience and interests and its relation to SAV employment.

4. Effect of experimental arrangement, experimenter effect: the interviewees may respond more favorably towards SAV techniques due to effect of leading questions from the interviewer. This may result in an information need data which does not reflect actual need. To counter this, the interview transcripts were studied carefully to determine whether there were leading questions.

5. Difference in treatment: since the data comes from different researchers, it is possible, that there could be a difference in treatment between different cases.

Possible external threats are as follows:

1. Population-related threats: a sample of each stakeholder type is moderate and contains from 3 to 5 people. There is no expectation for these findings to be generalized to describe requirements of a population. However, to mitigate associate problems, as much data as possible was analyzed in a short period of time, to represent different stakeholders from different cases.

In respect to construct validity, no considerable threats were identified.

In terms of reliability:

1. Replicability: this study is not dependent on particular researchers, and if a study was conducted on the same group of participants, the results should be the same.
2. Received data from cases 1-3: the received data from other researchers could be of concern, which would threated validity of results, however, it was received from trustworthy sources that have been previously validated.

D. Further research and Improvements

As it was mentioned before, a considerable improvement would be to acquire more data, particularly about developers' demands, in order to make the conclusions more generalizable.

Another possible step would be to display visualizations of a system using different techniques, such as graph-, notation-, matrix-, and metaphor – based techniques to a variety of stakeholders, and then conduct the interviews or distribute surveys aiming to find which techniques were most appropriate.

A possible further step could be to conduct cross-sectional studies to investigate relationships between a case's development practices' maturity, product's domain and state of SAV employment.

Lastly, effect of automatic tools employment on time required for system comprehension, newcomers training, communication, decision making and monitoring evolution of a system could be investigated for a longer periods of time. This would help to determine whether automatic SAV visualization could improve costs and quality of a development and demonstrate the value or lack of it of automatic SAV.

ACKNOWLEDGEMENTS

The authors of this paper would like to thank our supervisors, Truong Ho Quang and Michel Chaudron, as well as our industrial partners, who participated in the interviews.

REFERENCES

[1] Jyotri Priya, Sharif S. KB, Badri HS., *Software Architecture Visualization*, International Journal of Emerging Research in Management&Technology, ISSN: 2278-9350, 2013.

[2] Mattila, A.L., Ihantola, P., Kilamo, T., Luoto, A., Nurminen, M. and Väätäjä, H., 2016, October. Software visualization today: systematic literature review. In *Proceedings of the 20th International Academic Mindtrek Conference* (pp. 262-271). ACM.

[3] McGrath, M.B. and Brown, J.R., 2005. Visual learning for science and engineering. *IEEE Computer Graphics and Applications*, *25*(5), pp.56-63.

[4] ISO, M., 2011. *Systems and Software Engineering–Architecture Description* (pp. 1-46). ISO/IEC/IEEE 42010.

[5] Telea, A., Voinea, L. and Sassenburg, H., 2010. Visual tools for software architecture understanding: A stakeholder perspective. *IEEE software*, *27*(6), pp.46-53.

[6] Shahin, M., Liang, P. and Babar, M.A., 2014. A systematic review of software architecture visualization techniques. *Journal of Systems and Software*, *94*, pp.161-185.

[7] Gallagher, K., Hatch, A. and Munro, M., 2008. Software architecture visualization: an evaluation framework and its application. *IEEE Transactions on Software Engineering*, *34*(2), pp.260-270.

[8] Carpendale, S. and Ghanam, Y., 2008. *A survey paper on software architecture visualization*. University of Calgary.

[9] Cornelissen, B., Zaidman, A. and van Deursen, A., 2011. A controlled experiment for program comprehension through trace visualization. *IEEE Transactions on Software Engineering*, *37*(3), pp.341-355.

[10] Sjøberg, D.I., Hannay, J.E., Hansen, O., Kampenes, V.B., Karahasanovic, A., Liborg, N.K. and Rekdal, A.C., 2005. A survey of controlled experiments in software engineering. *IEEE transactions on software engineering*, *31*(9), pp.733-753.

[11] Sjoberg, D.I., Dyba, T. and Jorgensen, M., 2007, May. The future of empirical methods in software engineering research. In *Future of Software Engineering, 2007. FOSE'07* (pp. 358-378). IEEE.

[12] Eisenhardt, K.M., 1989. Building theories from case study research. *Academy of management review*, *14*(4), pp.532-550.

[13] Clarke, V. and Braun, V., 2014. Thematic analysis. In *Encyclopedia of critical psychology* (pp. 1947-1952). Springer New York.

[14] Zainal, Z., 2007. Case study as a research method. *Jurnal Kemanusiaan*, *9*.

[15] Yin, R.K., (1984). Case Study Research: Design and Methods. Beverly Hills, Calif: Sage Publications.

[16] Langelier, G., Sahraoui, H. and Poulin, P., 2005, November. Visualization-based analysis of quality for large-scale software systems. In *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering* (pp. 214-223). ACM.

[17] Sharafi, Z., 2011, June. A systematic analysis of software architecture visualization techniques. In *Program Comprehension (ICPC), 2011 IEEE 19th International Conference on* (pp. 254-257). IEEE.

[18] Condensation of class diagrams using machine learning by Filip Brynfor

[19] Wettel, R. and Lanza, M., 2007, June. Visualizing software systems as cities. In *Visualizing Software for Understanding and Analysis, 2007. VISSOFT 2007. 4th IEEE International Workshop on* (pp. 92-99). IEEE.

[20] Byelas, H. and Telea, A., 2006, September. Visualization of areas of interest in software architecture diagrams. In *Proceedings of the 2006 ACM symposium on Software visualization* (pp. 105-114). ACM.

[21] Panas, T., Epperly, T., Quinlan, D., Saebjornsen, A. and Vuduc, R., 2007, July. Communicating software architecture using a unified single-view visualization. In *Engineering Complex Computer Systems, 2007. 12th IEEE International Conference on* (pp. 217-228). IEEE.

[22] Chikofsky, E.J. and Cross, J.H., 1990. Reverse engineering and design recovery: A taxonomy. *IEEE software*, *7*(1), pp.13-17.

[23] Burden, H., Heldal, R. and Whittle, J., 2014, September. Comparing and contrasting model-driven engineering at three large companies. In *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (p. 14). ACM.

[24] Priya, J. and HS, M.S.K.B., Software Architecture Visualization.

[25] Balzer, M., Noack, A., Deussen, O. and Lewerentz, C., 2004. Software landscapes: Visualizing the structure of large software systems. In *IEEE TCVG*.

[26] Poort, E.R. and van Vliet, H., 2011, June. Architecting as a risk-and cost management discipline. In *Software Architecture (WICSA), 2011 9th Working IEEE/IFIP Conference on* (pp. 2-11). IEEE.

[27] McNair, A., German, D.M. and Weber-Jahnke, J., 2007, October. Visualizing software architecture evolution using change-sets. In *Reverse Engineering, 2007. WCRE 2007. 14th Working Conference on* (pp. 130-139). IEEE.

[28] Favre, J.M. and Cervantes, H., 2002. Visualization of component-based software. In *Visualizing Software for Understanding and Analysis, 2002. Proceedings. First International Workshop on* (pp. 51-60). IEEE.

[29] Hammad, M., Collard, M.L. and Maletic, J.I., 2010, June. Measuring class importance in the context of design evolution. In *Program Comprehension (ICPC), 2010 IEEE 18th International Conference on* (pp. 148-151). IEEE.

[30] Samia, M. and Leuschel, M., 2009, May. Towards pie tree visualization of graphs and large software architectures. In *Program Comprehension, 2009. ICPC'09. IEEE 17th International Conference on* (pp. 301-302). IEEE.

[31] Duszynski, S., Knodel, J. and Lindvall, M., 2009, March. Save: Software architecture visualization and evaluation. In *Software Maintenance and Reengineering, 2009. CSMR'09. 13th European Conference on* (pp. 323-324). IEEE.

[32] Wu, B.H., 2011, March. Let's enforce a simple visualization rule in software architecture. In *Information Science and Technology (ICIST), 2011 International Conference on* (pp. 427-433). IEEE.

[33] Merino, L., Ghafari, M. and Nierstrasz, O., 2016, October. Towards actionable visualisation in software development. In *Software Visualization (VISSOFT), 2016 IEEE Working Conference on* (pp. 61-70). IEEE.

[34] Kobayashi, K., Kamimura, M., Yano, K., Kato, K. and Matsuo, A., 2013, May. SArF map: Visualizing software architecture from feature and layer viewpoints. In *Program Comprehension (ICPC), 2013 IEEE 21st International Conference on* (pp. 43-52). IEEE.

[35] Grundy, J. and Hosking, J., 2000. High-level static and dynamic visualisation of software architectures. In *Visual Languages, 2000. Proceedings. 2000 IEEE International Symposium on* (pp. 5-12). IEEE.

[36] Knodel, J., Muthig, D. and Naab, M., 2006, October. Understanding Software Architectures by Visualization--An Experiment with Graphical Elements. In *Reverse Engineering, 2006. WCRE'06. 13th Working Conference on* (pp. 39-50). IEEE.

[37] Koschke, R., 2003. Software visualization in software maintenance, reverse engineering, and re-engineering: a research survey. *Journal of Software Maintenance and Evolution: Research and Practice*, *15*(2), pp.87-109.

[38] Khan, T., Barthel, H., Ebert, A. and Liggesmeyer, P., 2012. Visualization and evolution of software architectures. In *OASIcs-OpenAccess Series in Informatics* (Vol. 27). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[39] Holten, D., 2006. Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. *IEEE Transactions on visualization and computer graphics*, *12*(5), pp.741-748.

[40] Balzer, M. and Deussen, O., 2007, February. Level-of-detail visualization of clustered graph layouts. In *Visualization, 2007. APVIS'07. 2007 6th International Asia-Pacific Symposium on* (pp. 133-140). IEEE.

[41] Lungu, M. and Lanza, M., 2007, March. Exploring inter-module relationships in evolving software systems. In *Software Maintenance and Reengineering, 2007. CSMR'07. 11th European Conference on* (pp. 91-102). IEEE.

[42] Termeer, M., Lange, C.F., Telea, A. and Chaudron, M.R., 2005. Visual exploration of combined architectural and metric information. In *Visualizing Software for Understanding and Analysis, 2005. VISSOFT 2005. 3rd IEEE International Workshop on* (pp. 1-6). IEEE.

[43] LaToza, T.D. and Myers, B.A., 2010, October. Hard-to-answer questions about code. In *Evaluation and Usability of Programming Languages and Tools* (p. 8). ACM.

[44] Hofmeister, C., Nord, R.L. and Soni, D., 1999. Describing software architecture with UML. In *Software Architecture* (pp. 145-159). Springer US.

[45] ISO, I., 2011. IEEE: ISO/IEC/IEEE 42010: 2011: Systems and Software Engineering, Architecture Description. *Proceedings of Technical Report*.

[46] Nattapon, T. and Florence, M., 2016. Visualization of Electrical Architectures In the Automotive Domain Based on the Needs of Stakeholders.

[47] Brynfors, F., 2016. Condensation of class diagrams using machine learning.

APPENDIX

A. Interview Questions for Ericsson (Case 4)

1. Background questions
   - 1.1. What is your name? Which Department? How long have you been here?
   - 1.2. Do you have a title for your position?
   - 1.3. What are your main roles/tasks?
   - 1.4. Do you work in a team? How many people are there in your team? What role do you usually play in your team?
   - 1.5. Do you have any experiences with software design (CASE tools? UML?)
   - 1.6. How long have you been working with software design?

2. Software Design Process
   - 2.1. Can you briefly explain the software design process in the system that you are working with? Where are you involved in the process?
   - 2.2. Can you briefly describe one of your typical working days?

3. Existing SAV of the system
   - 3.1. Are you using any visualization about software architecture to support your work?
     - 3.1.1. If yes, please clarify in which context, which specific tasks?
     - 3.1.2. If yes, which do you like the most? Why?
     - 3.1.3. If no, what are the reasons for not using it?
     - 3.1.4. If no, do you have a mind map of the system? How does it look like?
   - 3.2. Do you find visualization useful?
   - 3.3. What methods of visualization are used?
   - 3.4. Does it provide the information that you need? What kind of information is it?
   - 3.5. What information is lacking? What is missing complementary visualization that could be used?
   - 3.6. Do you use any tools for visualization? What kind of tools? Are there lacking tools?

4. Different levels of abstraction
   - 4.1. Do you comprehend the system at different levels of details/abstractions? Can you explain why?
     - 4.1.1. If NOT, at which level of details that you like to see the system the most?