Accepted to the ACM International Conference on the Foundations of Software Engineering (FSE '24)

Supporting Early Architectural Decision-Making through Tradeoff Analysis: A Study with Volvo Cars

Karl Öqvist
Department of Computer Science and Engineering
Chalmers | University of Gothenburg
Gothenburg, Sweden
karloq@student.chalmers.se

Jacob Messinger
Department of Computer Science and Engineering
Chalmers | University of Gothenburg
Gothenburg, Sweden
mejacob@student.chalmers.se

Rebekka Wohlrab
Department of Computer Science and Engineering
Chalmers | University of Gothenburg
Gothenburg, Sweden
wohlrab@chalmers.se

ABSTRACT

As automotive companies increasingly move operations to the cloud, they need to carefully make architectural decisions. Currently, architectural decisions are made ad-hoc and depend on the experience of the involved architects. Recent research has proposed the use of data-driven techniques that help humans to understand complex design spaces and make thought-through decisions. This paper presents a design science study in which we explored the use of such techniques in collaboration with architects at Volvo Cars. We show how a software architecture can be simulated to make more principled design decisions and allow for architectural tradeoff analysis. Concretely, we apply machine learning-based techniques such as Principal Component Analysis, Decision Tree Learning, and clustering. Our findings show that the tradeoff analysis performed on the data from simulated architectures gave important insights into what the key tradeoffs are and what design decisions shall be taken early on to arrive at a high-quality architecture.

CCS CONCEPTS

• **Software and its engineering** → *Extra-functional properties*; **Software architectures**; *Cloud computing*; *Abstraction, modeling and modularity*; *Software design tradeoffs.*

KEYWORDS

software architecture, architectural analysis, cloud systems, design decisions, principal component analysis, tradeoff analysis

ACM Reference Format:
Karl Öqvist, Jacob Messinger, and Rebekka Wohlrab. 2024. Supporting Early Architectural Decision-Making through Tradeoff Analysis: A Study with Volvo Cars. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering (FSE Companion '24), July 15–19, 2024, Porto de Galinhas, Brazil.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3663529.3663860

1 INTRODUCTION

Modern cars are commonly made up of complex software systems that continuously record and process data on the cloud [21]. To ensure that the right levels of quality are achieved, their architectures must be carefully designed, e.g., with respect to performance, availability, and cost [18]. Decisions taken in the design phase of software architectures can greatly impact the system's quality [12]. However, design decisions are often made in an ad-hoc manner rather than being data-driven with concrete evidence for how the system will perform and what factors influence the system's quality [2, 11]. One concern when designing software architectures are tradeoffs, which occur when an improvement in one area entails a negative impact on another area [10]. Most decisions taken in the design phase come with inadvertent tradeoffs. An example tradeoff arises from using more powerful hardware that leads to greater performance but will also increase the monetary cost of the system.

Researchers have investigated architectural decision-making in the design phase [20]. In recent years, a trend has been to postpone architectural decisions as much as possible. At the same time, some decisions are important to be made at the beginning of the development—especially those that are hard to isolate and for which a change would have a large impact on the rest of the architecture. Identifying these decisions is crucial and the focus of our approach. Those decisions often relate to tradeoffs in the architecture.

Past research has proposed to use machine learning techniques to explain quality tradeoffs [3, 4]. However, to the best of our knowledge, such tradeoff analysis techniques have not been applied to architectural decision-making in an industrial setting with real-world challenges. In this paper, we explore this issue and aim to answer the following questions: To what extent is it feasible to understand architectural tradeoffs based on simulation data? What are key decisions that are important to be made at the beginning of the development? How can tradeoffs be effectively visualized? How understandable are such visualizations?

In this paper, we present a study that focuses on developing an approach to simulate and elicit architectural tradeoffs in the automotive domain. The approach assumes that architects have thought of potential architecture candidates and want to better assess the quality tradeoffs that these candidates come with. Moreover, they want to analyze the impact of both high- and low-level decisions. We collaborated with a team of architects aiming to redesign their architecture and analyze what decisions to make early on.

2 SCENARIO USED IN THIS STUDY

A typical architectural problem in the design phase at Volvo Cars has been selected as our example scenario. The scenario was used for all modeling, simulation, and analysis. The wanted architecture is intended for a system that is responsible for delivering diagnostic

FSE Companion '24, July 15–19, 2024, Porto de Galinhas, Brazil

ACM ISBN 979-8-4007-0658-5/24/07
https://doi.org/10.1145/3663529.3663860

data from the car's battery and contains the following steps: (1) The vehicle produces 8 kB of diagnostic data with a frequency of 1 Hz. (2) The data is stored in a local cache that can be set up to act as a batching component for the data. It only sends data to the cloud when the cache contents have reached a certain threshold size. (3) The data is transported to the cloud. (4) The data undergoes computations to calculate important metrics such as time to empty and charge rate. (5) The data is stored in a cloud storage. To realize the cloud side of the architecture, Amazon Web Services (AWS) will be used. Within the AWS ecosystem, many services exist and many choices can be made that have an impact on software architecture.

3 RELATED WORK

A literature review on architecture evaluation in high-uncertainty environments [19] concluded that utilizing machine learning techniques when evaluating architectures significantly improves the decision-making process. However, only 25% of the employed runtime evaluations use machine learning approaches [19]. Our approach employs machine learning techniques to explain the architectural alternatives that architects choose from. Previous architecture optimization approaches have mostly focused on generating architectural configurations without supporting mechanisms that help people understand the underlying tradeoffs [1].

Previous research has focused on understanding architectural tradeoff spaces, which explores the possibility of linking quality attributes to architectural design variables using machine learning techniques such as PCA [3, 5, 22]. The output of tools to generate architectural designs is often not easily digestible and understandable for practitioners, and the results are filled with noise that obscures the understanding of relations between variables [3]. This raises the need for techniques that guide architects in architectural decision-making. To address this challenge, related work has explored the use of radar charts and visualization to help humans understand architectural tradeoffs [6, 13]. To the best of our knowledge, none of those previous techniques have been applied and evaluated in the context of architectural decision-making in an industrial setting.

4 RESEARCH METHOD

This study was conducted using design science [7] in which we iteratively developed an approach for simulating cloud-based software architectures in order to elicit and explain architectural tradeoffs.

We included all participants involved with cloud-based software architectures in the company: (1) one software architect and (2) one cloud architect, both with more than 20 years of experience with software engineering; (3) a software architect with 10 years of experience, (4) an advanced engineering lead with 3 years of experience, and (5) a solution architect with more than 25 years of experience. We performed a series of interviews and focus groups.

We relied on three iterations, each spanning five weeks. An overview figure of what the iterations contained, what was added to the approach in each iteration, and how the approach was evaluated in between iterations can be found on Figshare[1], along with other supplementary material.

[1] https://doi.org/10.6084/m9.figshare.24258439.v2

5 CHALLENGES WITH TRADEOFF ANALYSIS

To understand practitioners' challenges, we performed five interviews of 20–30 minutes. In the following, we present the key challenges that should be addressed by a tradeoff analysis approach.

Time-consuming and non-standardized exploration of the design space. One finding throughout the interviews is that there is no standardized architecting process. Usually, several architects collaborate until a consensus is reached. Other than sketching ideas with a visualization tool and discussing them, there were no additional methods that the architects used for the evaluation and analysis of proposed solutions. The architects also stressed that they could not evaluate every possible solution and idea, which is why it is important to be able to quickly assess solutions and their tradeoffs.

Our approach requires architects to have deduced several different architecture candidates. The interviewees had many design ideas that seemed suitable for the scenario of the diagnostic data from the vehicles. The architects stated that they could not evaluate every possible topological setup and idea, which is why it is important to be able to quickly assess candidates and their tradeoffs.

Side-effects of decisions and complexity of problem space. In our interviews, it was found that it was important that a chosen solution does not have previously unidentified side effects when run in production. To mitigate this, it is crucial to understand the impact of architectural decisions. It was also stated that underestimating such impacts is a commonly made oversight and very difficult to completely avoid. The reason is that there are many decisions to make for all possible solutions, and controlling and analyzing them all would be too costly and intricate. One architect among the interviewees stated: "*I don't think anyone at the company has a large-scale view of the system. Meaning that we can make a little decision here, which adds a lot of complexity for someone else without us noticing, and vice-versa.*"

Reliance on individuals' experience levels. In the interviews, it was found that domain knowledge and previous experience are crucial for understanding architectural tradeoffs. Appropriate documentation of the services and components within the architecture was seen as essential. One interviewee additionally stated: "*When you are not so experienced with a concept, it is easy to focus too much on the advantages but not maybe anticipate the challenges.*"

6 APPROACH FOR DECISION SUPPORT

The models and simulations were developed using Simulink [14], a software used for graphical modeling and simulation of dynamic systems. The subsequent analysis and visualizations were developed in Python using a variety of libraries [8, 9, 15–17].

Fig. 1 shows the approach that was developed in this study. It goes from architecture candidates down to how analysis methods are applied to support decision-making. First, the proposed architecture is modeled as a set of components and connections in Simulink (1). Each component models a real-world architectural component or service. Components are connected similarly to how they would in the real world. By exploring multiple combinations of parameters and other architectural decisions, a thorough exploration of the design space is enabled. In (2), a Matlab script acts as a guide to how a modeled architecture gets loaded with parameter values.

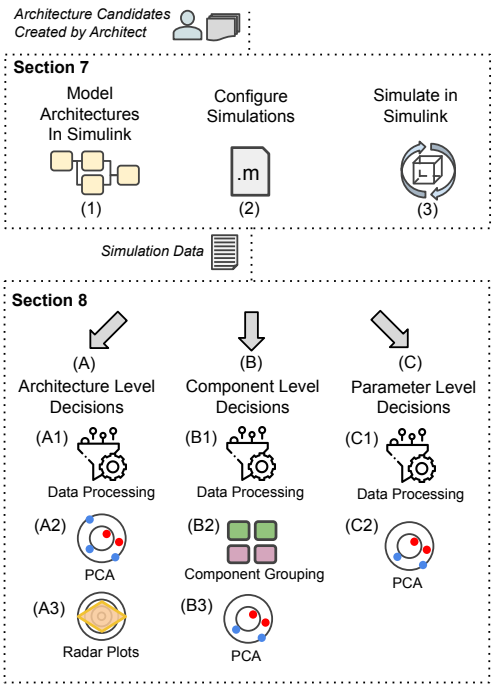

Figure 1: Our approach for design decision support through simulations and tradeoff analysis

The script also configures the data input and maximum duration time of the simulation. For this study, the script configures the simulation to simulate 15 minutes of constant data upload from 1000 vehicles. Step (3) is to run the simulations. For this study, a total of 12136 simulations were run, each representing a unique architectural setup. The total simulation time was approximately 4 hours using an 8-core CPU with 128GB of RAM.

The output of the simulations is one data frame containing: the simulated architecture, parameter values, components in use, and the quality metrics of cost, latency, load sensitivity, and complexity. The data frame used in this study can be found on Figshare[1] and Github, along with the simulation data and analysis code[2].

The next action in this approach is data analysis and visualization. It is possible to apply a multitude of techniques to be able to support a broad variety of design decisions and are thus subject to change based on the needs of the practitioners. The three paths taken in this paper (A, B, C in the figure) showcase the analyses deemed helpful by the architects at Volvo Cars. In the first steps (A1), (B1), (C1), the data frame is filtered using data frame slicing tailored to what kind of questions we want to answer. A subsection of Pareto-optimal data samples is retrieved. Exceptionally bad or good setups are filtered out to capture the general trends of each architecture. The later steps are analysis and visualization tools, where (A2), (B3), and (C2) denote Principal Component Analysis (PCA). (A3) denotes radar plots and (B2) relies on decision flow charts.

[2]https://github.com/karloq/architectural-tradeoff-analysis

7 MODELING CLOUD ARCHITECTURES

In total, seven architecture candidates were discussed at Volvo. They belong to three design ideas, named `Simple`, `Stream`, and `Sophisticated`. On Figshare[1], a table of all possible parameters, their value ranges, and a description of their effect can be found.

Simple. This solution uses either *Kinesis* or *IoT Core* for data transport and *Lambda* or *Fargate* as processing services. If *Lambda* is used for processing, *SQS* is used as an intermediary service before *Lambda*, to utilize its resources better and optimize total computation time. The design uses a limited caching capability, which is why in reality the software code would have to be developed with great care to ensure that data is not lost if services were to crash.

Stream. This design idea also uses *Kinesis* or *IoT Core* for data transport. The data then gets ingested into *DynamoDB*, where a *Lambda* instance keeps track of how much data has been received from each car. When ample data has been received, it sends the batched data as a job to *SQS* that acts as a queue. The jobs are then fetched by either *Fargate* or *Lambda* for processing. This offers a solution to some of the concerns of the *Simple* architecture because both *DynamoDB* and the *SQS* act as a caching service that can be configured to re-transmit data if the processing services *Lambda* and *Fargate* were to lose data for some reason. The data gets batched into processing jobs based on what car they stem from. That ensures that data from the same vehicle is not processed in parallel, which makes it much easier to ensure stricter data consistency.

Sophisticated. This design idea uses *IoT Core* for data transport. It then gets processed by a *Lambda* function that checks against a *DynamoDB* database if the container for the data processing is active. All cars are split up so that they have a specific container for their processing. If active, the data is sent to the container. If not active, the data gets stored in an *SQS* queue while the container gets started by a *StepFunction* service. When it is started, the data in the queue is sent to the container, and *DynamoDB* is altered to indicate that the container is active so that new data to that container is sent directly. *DynamoDB* and *SQS* can thus re-transmit data and act like caches. Moreover, individual containers can run on different cars. The containers are preassigned to the cars beforehand and do not necessarily need to be replicas of each other, since *EC2* can run different containers in the same service. This cannot be done in `Simple` and `Stream` as instances in *Lambda* and *Fargate* must be replicas of each other. The Sophisticated design idea might prove beneficial for worldwide operations since it enables different data processing based on the region the data stems from.

Quality metrics. Performance is measured by the total latency caused by the blocks divided by the number of data packets from the cars that have been computed. *Cost* is measured by the price of running the system for the simulation time and was modeled based on the pricing information on the Amazon website. *Complexity* measures how many parameters and blocks a setup contains (not counting *Source*). *Load sensitivity* indicates how sensitive the cost and latency of the system are when the load increases tenfold. A Load Sensitivity value of 10 means that the Cost and Latency increase tenfold, i.e., it scales linearly. If the Load Sensitivity is 0, the Cost and Latency are constant when increasing the load.

8 FINDINGS FROM TRADEOFF ANALYSIS

8.1 Findings on Architecture Level Decisions

In early stages, architects typically ask themselves: *What tradeoffs do different architectural candidates entail?* This question is a part of understanding how architectural decisions affect quality. The simulation data of 1200 configurations is useful since factual evidence is needed to answer such questions. To make use of this multivariate simulation data, it needs to be broken down.

Fig. 2 shows clusters of four of the seven architecture candidates. The radar plots are created by computing mean values of the quality measures for each design idea and normalizing the scales. What can be seen in Fig. 2 is that there exist two general, somewhat skewed, shapes: a kite, as can be seen for `Stream_1` and `Simple_1`, and a trapezoid, as can be seen for `Sophisticated_1` and `Simple_3`. The existence of such shapes shows that no architecture minimizes all qualities. Tradeoffs always have to be made. Stream configurations are expensive and complex, but have good latency and are somewhat load-sensitive. Simple configurations are rather cheap and not complex, but can be very load-sensitive. Sophisticated configurations come with a high latency and load sensitivity, but are neither very complex nor costly. These findings can help architects understand what tradeoffs come with different configurations.

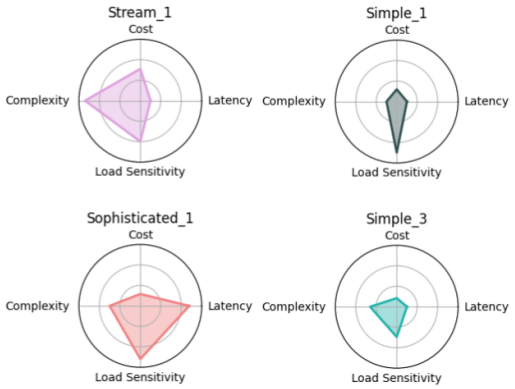

Figure 2: Radar plots showing 4 architectures with their quality measures, with high values (meaning poor quality metric for the attribute) closer to the outer rim of the circle.

8.2 Findings on Component Level Decisions

To better understand the tradeoffs, architects might ask: *What specific components and services impact the qualities and cause tradeoffs?*

In Fig. 3, a decision flow chart is depicted. Each architecture can be seen as making five key decisions, i.e., regarding *Source*, *Transport*, *Intermediary*, *Processing*, and *Storage*. In each step, different decisions can be made. For *Source*, architects can choose between *Caching* and *No Caching*. Caching is positively correlated with all quality measures, whereas no caching is negatively correlated with them. It can be seen that the main cause for high *Cost* and *Latency* is the decision not to cache data in the car and the use of MQTT as the transport protocol. What can also be seen is that all *Intermediaries* entail a tradeoff either through lowering the cost but increasing the

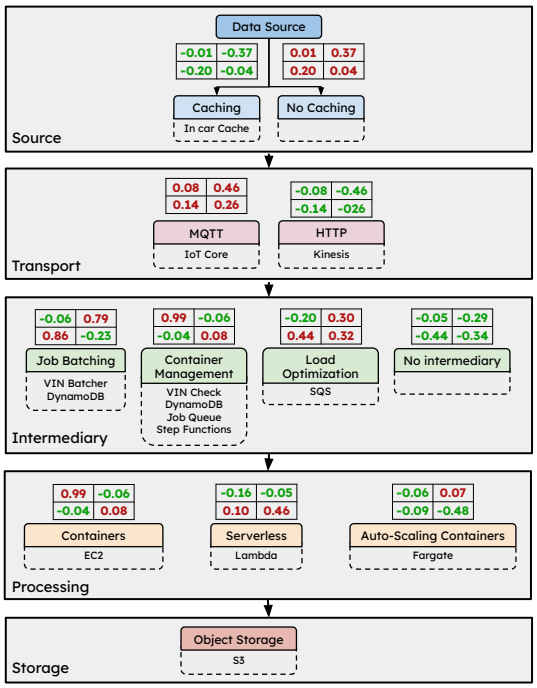

Figure 3: Decision Flow chart describing design decisions and their quality attribute correlations (top left = Latency, top right = Cost, bottom left = Complexity, bottom right = Load Sensitivity). Positive numbers indicate positive correlations and negative numbers indicate negative correlations.

latency, or vice-versa. This gives further insight into the reasons behind why `Simple` and `Stream` outperform `Sophisticated`: they omit the *Container Management* and *Containers* features, which are the features that have the highest correlation with high latency.

8.3 Findings on Parameter Level Decisions

Many of the AWS components that are considered by Volvo Cars can be modified and configured with the use of settings and parameters. The differences in architectural setups and the selection of parameters have a large impact on how certain components function and thus change the architecture significantly. It therefore exists a need to be able to analyze the components on a more fine-granular level to see how these parameters and settings affect the quality metrics. We aim at answering questions like: *What parameters are important for the impact on the quality attributes?*

To answer the question, PCA is applied, which is a widely used statistical method that aims to reduce the dimensionality of data. The approach divides the data into Principal Components (PC), which are linear combinations of the variables in the data. Fig. 4 shows the results of a PCA performed on the top 15% of the samples from each architecture candidate in the simulation data. The variances that PC1 and PC2 explain are expressed as a percentage on the corresponding axes. The plot describes 86.2% of the variance in the data. Points that are located close to each other are positively correlated, whereas points that are located on opposite ends

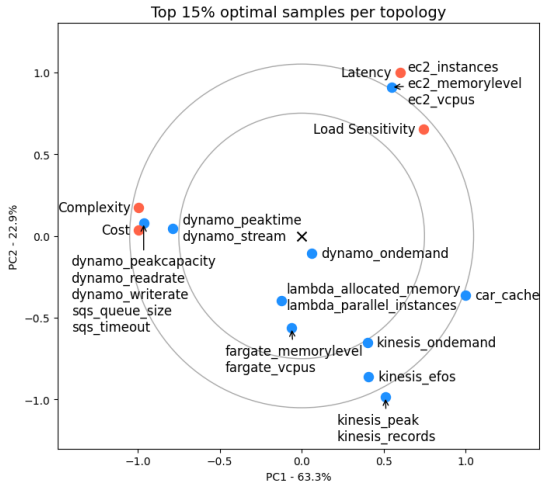

Figure 4: PCA plot depicting the parameters and quality attributes for the top 15% of the samples

of the PCA plot are negatively correlated. What it shows is, first, that all quality variables lie far from the origin (marked by an X in the figure), which means that they are relevant for explaining the variance in the data. Secondly, all parameters that stem from the same component are tightly clustered together. That makes sense since if a component does not exist in an architecture, all its parameter values will be zero. Therefore, this PCA plot can help to explain how the existence of components affects the quality of the modeled system. It can be seen that EC2 parameters have a significant positive correlation with load sensitivity and latency. Similarly, Dynamo and SQS parameters strongly correlate with cost and complexity. On the contrary, all parameters in the lower right quadrant, i.e., car cache, and Kinesis parameters are negatively correlated with cost and complexity. These insights are useful as they indicate what tradeoffs to expect depending on the chosen parameter values.

8.4 Qualitative Evaluation

The participating architects found the PCA loading plots useful because they gave them a good indication of the quality tradeoffs of each architectural candidate. They liked the fact that so much information about the general trends in the data could be discerned from the plots. They also acknowledged the fact that this is something far off any practice they employ today. The big disadvantage with the PCA loading plots which was confirmed in the evaluation is that they are hard to analyze with little experience. Therefore, the radar plots were seen as beneficial to complement PCA plots and also as a better option for quick comparisons.

The participants found the approach useful for early decision-making and analysis. The visualizations sparked a discussion among the architects regarding the actual architectural scenario. They concluded that they conveyed the tradeoffs the architect team had experienced in previous architectures they designed, but had not been thinking of before it was deployed in a real demo. The team stated that the approach really can be used in their everyday work, as long as they can trust the simulation data and its correctness.

9 THREATS TO VALIDITY

Internal Validity: While we focused on the main design decisions and quality attributes, some factors might not have been considered that might change the overall results. Close collaboration with architects at Volvo Cars was instrumental in ensuring that the modeled system behaves realistically. Any specific values used in this analysis are not to be used as factual statements.

External Validity: Because the nature of this study is specific to the selected scenario, external validity is low. Generalizability was not the goal. Instead, we investigated one particular real-world case in depth. We aimed to describe the case context in detail, so that others can consider generalization based on analogy.

Reliability: We made our interview guide, the approach's code, and our example data available. Many of the visualization techniques used utilize open-source libraries, so we encourage others to apply them to further example scenarios and systems.

Construct Validity: Constructs such as "tradeoff" and "design space" might be understood in different ways. Explaining these to the interviewees helped us improve construct validity.

10 CONCLUSIONS

Currently, architects are often left to ad-hoc decision-making based on experience and discussions. In this paper, we investigated to what extent architectural tradeoffs can be explored using a data-driven approach. The results of our research highlight the value of visualization techniques [3, 5, 13, 22], such as PCA plots, radar plots, and decision trees, in supporting architectural decision-making.

The approach assumes that ideas for potential architecture candidates exist whose tradeoffs and parameter values architects want to investigate. It took us one full-time person week to set up the scripts and create the first model for `Simple_1`. The next models could be created within a few hours, and the plots used in this paper could be developed and generated within a few minutes. We conclude that while the approach does require an initial time investment, it makes it easier to explore architectural alternatives later on. The techniques employed in this study enabled a fine-granular analysis of tradeoffs, key architectural decisions, and parameter selection.

We chose one scenario, for which we described our findings in depth. Applying the approach in other scenarios will help us to assess the general usefulness. In this work, we focused on latency, cost, complexity, and load sensitivity as quality metrics. It will be interesting to assess how other quality concerns can be modeled.

An interesting aspect to investigate in future work is the extent to which providing the input for the simulation design itself already supports systematic decision-making on the architect's side—and/or how much of the insights come through the visualizations and discussions.

ACKNOWLEDGMENTS

The authors would like to thank the participants from Volvo Cars for their support of the study. This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

REFERENCES

[1] Aldeida Aleti, Barbora Buhnova, Lars Grunske, Anne Koziolek, and Indika Meedeniya. 2013. Software Architecture Optimization Methods: A Systematic Literature Review. *IEEE Transactions on Software Engineering* 39, 5 (2013), 658–683. https://doi.org/10.1109/TSE.2012.64

[2] P. Bengtsson and J. Bosch. 1998. Scenario-based software architecture reengineering. In *Proc. of the Fifth International Conference on Software Reuse*. IEEE, 308–317. https://doi.org/10.1109/ICSR.1998.685756

[3] Javier Cámara, Rebekka Wohlrab, David Garlan, and Bradley Schmerl. 2023. ExTrA: Explaining architectural design tradeoff spaces via dimensionality reduction. *Journal of Systems and Software* 198 (2023), 111578. https://doi.org/10.1016/j.jss.2022.111578

[4] Javier Cámara, Rebekka Wohlrab, David Garlan, and Bradley Schmerl. 2024. Focusing on What Matters: Explaining Quality Tradeoffs in Software-Intensive Systems Via Dimensionality Reduction. *IEEE Software* 41, 1 (2024), 64–73. https://doi.org/10.1109/MS.2023.3320689

[5] J. Andres Diaz-Pace, Rebekka Wohlrab, and David Garlan. 2023. Supporting the Exploration of Quality Attribute Tradeoffs in Large Design Spaces. In *Proc. of the European Conference on Software Architecture*, Bedir Tekinerdogan, Catia Trubiani, Chouki Tibermacine, Patrizia Scandurra, and Carlos E. Cuesta (Eds.). Springer Nature Switzerland, Cham, 3–19.

[6] Sebastian Frank and André van Hoorn. 2020. SQuAT-Vis: Visualization and Interaction in Software Architecture Optimization. In *Proc. of the European Conference on Software Architecture (ECSA)*, Vol. 1269. Springer, 107–119. https://doi.org/10.1007/978-3-030-59155-7_9

[7] Alan R Hevner. 2007. A Three Cycle View of Design Science Research. *Scandinavian Journal of Information Systems* 19 (2007). Issue 2.

[8] J. D. Hunter. 2007. Matplotlib: A 2D graphics environment. *Computing in Science & Engineering* 9, 3 (2007), 90–95. https://doi.org/10.1109/MCSE.2007.55

[9] Plotly Technologies Inc. 2015. *Collaborative data science*. Montreal, QC. https://plot.ly

[10] Ton Kostelijk. 2005. Misleading architecting tradeoffs [DVD hard-disk architecture tradeoff analysis method]. *Computer* 38, 5 (2005), 20–26. https://doi.org/10.1109/MC.2005.165

[11] Philippe Kruchten, Rafael Capilla, and Juan Carlos Duenas. 2009. The decision view's role in software architecture practice. *IEEE Software* 26, 2 (2009), 36–42. https://doi.org/10.1109/MS.2009.52

[12] Francisca Losavio, Ledis Chirinos, Nicole Lévy, and Amar Ramdane-Cherif. 2003. Quality Characteristics for Software Architecture. *The Journal of Object Technology* 2 (2003), 133. Issue 2. https://doi.org/10.5381/jot.2003.2.2.a2

[13] Jason Mashinchi and Javier Cámara. 2020. Voyager: Software Architecture Tradeoff Explorer. In *Proc. of the European Conference on Software Architecture (ECSA)*, Vol. 1269. Springer, 55–67. https://doi.org/10.1007/978-3-030-59155-7_5

[14] MathWorks. [n. d.]. Simulink - Simulation and Model-Based Design. https://se.mathworks.com/help/simulink/, accessed 2023-05-09.

[15] Wes McKinney. 2012. *Python for data analysis: Data wrangling with Pandas, NumPy, and IPython*. O'Reilly Media, Inc.

[16] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830. https://doi.org/10.5555/1953048.2078195

[17] pypi.org. 2024. paretoset. https://pypi.org/project/paretoset, accessed 2024-05-09.

[18] Sofia Sherman and Naomi Unkelos-Shpigel. 2014. What do software architects think they (should) do? Research in progress. In *Proc. of the CAiSE 2014 International Workshops*. Springer, 219–225. https://doi.org/10.1007/978-3-319-07869-4_20

[19] Dalia Sobhy, Rami Bahsoon, Leandro Minku, and Rick Kazman. 2021. Evaluation of software architectures under uncertainty: a systematic literature review. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 30, 4 (2021), 1–50. https://doi.org/10.1145/3464305

[20] Antony Tang, Maryam Razavian, Barbara Paech, and Tom-Michael Hesse. 2017. Human aspects in software architecture decision making: a literature review. In *Proc. of the IEEE International Conference on Software Architecture (ICSA)*. IEEE, 107–116. https://doi.org/10.1109/ICSA.2017.15

[21] Haoxin Wang, Tingting Liu, BaekGyu Kim, Chung-Wei Lin, Shinichi Shiraishi, Jiang Xie, and Zhu Han. 2020. Architectural design alternatives based on cloud/edge/fog computing for connected vehicles. *IEEE Communications Surveys & Tutorials* 22, 4 (2020), 2349–2377. https://doi.org/10.1109/COMST.2020.3020854

[22] Rebekka Wohlrab, Javier Cámara, David Garlan, and Bradley Schmerl. 2023. Explaining quality attribute tradeoffs in automated planning for self-adaptive systems. *Journal of Systems and Software* 198 (2023). https://doi.org/10.1016/j.jss.2022.111538

Received 2024-02-08; accepted 2024-04-18