Making Architecture Optimization Transparent with Tactic-Based Explanations

J. Andres Diaz-Pace
ISISTAN, CONICET and UNICEN University
Tandil, Buenos Aires, Argentina
andres.diazpace@isistan.unicen.edu.ar

David Garlan
School of Computer Science, Carnegie Mellon University
Pittsburgh, PA, USA
garlan@cs.cmu.edu

***Abstract*—Over the past decade, a number of automated techniques and tools have been developed for optimizing architectural designs with respect to quality-attribute goals. In these systems, the optimization process is typically seen as a black box, since it is not possible for a human to have access to the decisions that led to a particular solution generated by an optimization tool. Even when these decisions are available for inspection, the amount of information can be overwhelming for the architect. As a result, humans might not completely understand the rationale behind a given solution or trust that a tool made correct decisions. To mitigate this problem, we propose a semi-automated approach for generating textual explanations for any architectural solution produced by a tool. This kind of explanation provides a summary of the key architectural tactics that were applied to achieve an optimized architecture that satisfies a set of quality-attribute objectives. In this paper, we discuss two procedures for determining the key tactics to be explained. As an initial experiment, we used a popular optimization tool to generate solutions and explanations for a small but non-trivial design space involving performance, reliability, and cost objectives. We also performed an exploratory user study to assess the effectiveness of these explanations.**

***Index Terms*—architecture optimization, tactics, explainability, tool support, user study.**

I. Introduction

Designing a software architecture to meet its main requirements is a complex and frequently error-prone process. This process usually involves addressing a set of quality-attribute objectives (e.g., performance, reliability, or cost, among others) which might trade off with each other. Human architects normally develop a number of candidate solutions and evaluate them against the objectives. Each candidate is backed up by a design rationale that captures the *key decisions* (e.g., patterns, tactics, or technology choices) with their pros and cons, so that stakeholders can understand the solution. Given the complexity of today's systems, the space of possible architectural candidates for a set of quality attributes is often beyond human capabilities. In response, several automated tools for assisting architecture exploration via automated search and optimization have been developed over the last years [1] [2] [3].

Architecture optimization tools are good at performing efficient multi-objective search of the design space and returning to a human architect a set of design candidates with different quality-attribute trade-offs. Internally, these tools might rely on a variety of techniques (e.g., genetic algorithms, model checking, or hill climbing) to drive the search, and assemble possible solutions. For instance, a tool can automatically apply a sequence of transformations on an initial architecture for progressively improving one or more quality objectives, while considering various constraints and preferences. A tool can also employ sophisticated quality-attribute analyses (such as simulation) to assess each candidate. However, the design and analysis mechanisms used by the tool tend to be opaque to the architect, as they are typically fine-grained and overly-detailed for human comprehension. Instead, humans are most likely to reason about the solutions in terms of high-level constructs such as the key architectural patterns or design decisions, which are standard concepts in the architect's mindset.

In this context, there is therefore a mismatch between the *human model* driven by architecting concerns and the *tool model* driven by optimization concerns. Human-like explanations are required for how a generated solution came to have particular design characteristics or meet a quality-attribute objective. If a solution is not understood or trusted by architects or developers, they might ignore the tool output or carry out an implementation that fails to respect the generated design. These issues emphasize the need for explainability in software architecture and pose challenges such as:

- How to identify the key decisions of an architectural design being automatically generated by a tool?
- How to present those decisions to a human (e.g., in textual or graphical formats)?
- How to determine the level of detail that is appropriate for these explanations?

Driven mainly by the first question, we propose a semi-automated approach for creating textual explanations that summarize the key design decisions and their quality-attribute effects for a generated architecture. These explanations are based on the notion of *architectural tactics*, as design decisions that bridge a tool's optimization mechanisms and an architect's comprehension needs. On one hand, tactics can serve a tool to enable architectural transformations for improving an architecture. On the other hand, tactics can serve humans as anchors to answer questions about the outputs of that tool.

We have developed a prototype to demonstrate the ideas above, using two alternative procedures for generating explanations for a small design space that involves three quality-attribute objectives along with tactics for improving them. Furthermore, we conducted a small user study to assess the

2768-4288/22/$31.00 ©2022 IEEE
DOI 10.1109/ICSA-C54293.2022.00008

effectiveness of the tactic-based explanations when architects are exposed to design questions.

The rest of the paper is structured as follows. Section 2 motivates the need for human architects to understand the outputs of an optimization tool in terms of key decisions. In Section 3, we present our approach for generating tactic-based explanations with two procedures. Section 4 reports on the findings of our initial evaluation of the approach. Finally, Section 5 gives the conclusions and outlines future work.

II. Interpreting Architecture Optimization

An architecture optimization tool can be seen as a decision-making agent that generates one or more candidate architectures for improving an initial architecture with respect to a set of (possibly competing) quality-attribute objectives. In this setting, each candidate results from applying a sequence of transformations corresponding to design decisions made by the agent. This sequence is referred to as the agent's *strategy*. In existing approaches to architecture optimization, like PerOpteryx [4] or GATSE [3], the details of the strategy are opaque to the architect who only sees the process outputs (once a strategy has been chosen). Related works on architectural decisions have captured those decisions via knowledge management techniques [5] or templates[1]. Decision traceability has also been considered in prior research. However, less attention has been paid to interpretability or transparency aspects when humans analyze those decisions.

Fig. 1 illustrates an optimization scenario from the architect's external perspective. The architect provides an initial architecture A_0 along with three quality-attribute objectives that it should satisfy. Satisfaction is expressed in terms of predefined thresholds for the objectives. After an internal exploration process, the tool returns a candidate architecture A_n that meets the performance objective. At this point, and without knowing the strategy, a typical question posed by the architect is: ***why*** *does architecture A_n satisfy quality-attribute objective X?* For instance, she might want to know why the performance objective was met by A_n (and why cost and reliability were not). The *why-question* tries to understand both the architecture and the rationale that led to it. Other types of questions (e.g., why-not, how, what-if) could also be made [6].

To understand the role of tactics, let us depart from an initial client-server architecture (A_0) and a possible candidate (A_n) returned by the tool, as shown in Figs. 2 and 3 respectively. In this system, user requests ($R1$) pass through a load balancer that assigns the requests to one of the active service instances for processing. Each device is assumed to have a capacity to host up to six service instances. Let us assume that we have two simple tactics: `increaseCapacity(?device)` and `decreaseCapacity(?device)`, which refer to deploying service instances on predefined server devices. Each tactic has a different effect on the performance, reliability and cost properties of the design solution. For instance, having more service instances running concurrently (either on the same or different devices) improves latency and throughput.

[1]https://adr.github.io/

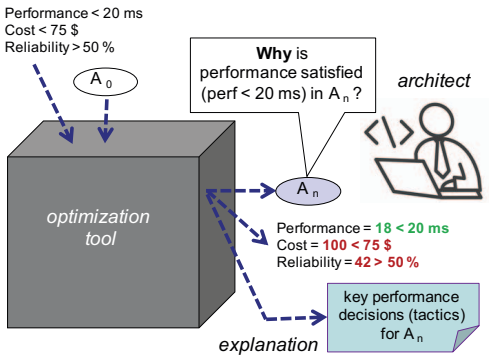

Fig. 1. Opaque architecture optimization and the role of explanations.

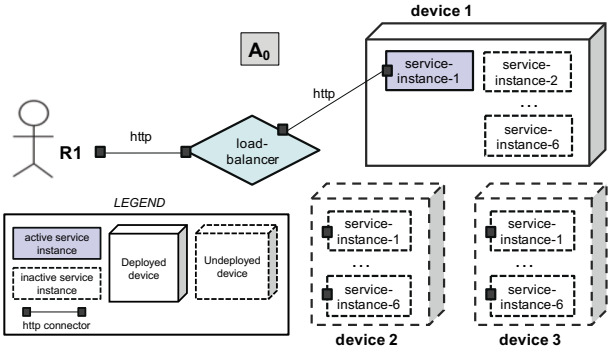

Fig. 2. Initial client-server architecture.

However, if a group of service instances runs on the same device, a failure in that device reduces the system processing capabilities. In addition, increasing the number of devices with active services uses more computational resources (if new servers need to be provisioned), and incurs additional costs. In this optimization scenario, the tool should apply the available tactics judiciously to balance the three objectives. To compute the effects of each tactic, we assume the tool relies on analysis models for performance, reliability and cost [4].

For candidate A_n the tool strategy entails a sequence of four architectural transformations (or tactic instances): `<increaseCapacity(device2), increaseCapacity(device2),increaseCapacity(device3),increaseCapacity(device4)>`. Fig. 4 shows the evolution of the quality-attribute properties of the intermediate architectures while reaching A_n. Although this is a simple example, in general, the number of intermediate and candidate architectures assessed by the tool can be large, and the strategy for arriving at a candidate involves a large number of tactic applications and quality-attribute analyses.

In the example, the tool could make the design strategy evident to the architect by exposing the complete tactic sequence for A_n. However, tactic sequences become lengthy even for relatively small design spaces (e.g., up to 10 tactics in our example), which causes architects to experience information overload. To deal with this problem, the tool could offer instead a summary with the most relevant tactics of the sequence for the *why-question* under analysis. The intuition here is that other (subsidiary) decisions will fol-

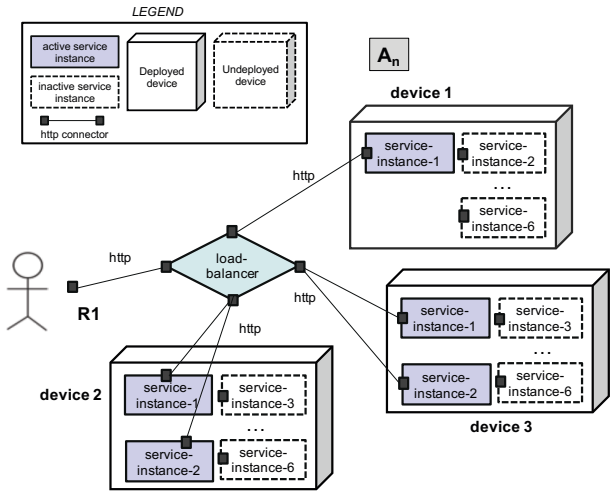

Fig. 3. Candidate architecture.

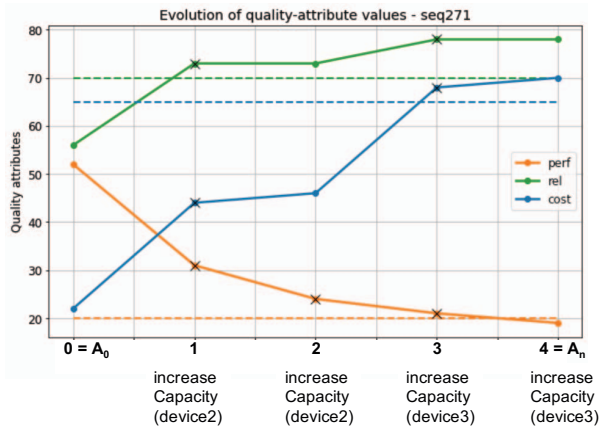

Fig. 4. Improvement of quality-attribute values over time.

low from the most relevant ones, or that certain decisions do not affect the quality attribute targeted by the question. Specifically, each tactic can be assigned a relevance score according to its support for the quality-attribute objective. For instance, a performance-oriented summary would include only `increaseCapacity(device2)` (at step 1), since this tactic produced the largest reduction in response time in the sequence of Fig. 4. Analogously, the cost summary would include `<increaseCapacity(device2), increaseCapacity(device3)>` because of their contributions to the final cost. We see these kinds of summaries as a local, human-oriented interpretation [7] of the optimization process, providing a simplified (or approximate) view of the tool's strategy for an architecture candidate.

III. A Tactic-based Explanation Approach

Before delving into the generation of explanations, we present the basic notions of our architecture optimization framework, which derive from a multi-objective optimization formulation [8]. Let us assume an architecture space composed of multiple architectures $AS = \{A_0, A_1, A_2, ..., A_n\}$, in which A_0 is designated as the initial architecture that needs to be improved. The specific techniques (e.g., [1], [4]) for exploring and populating AS are out of the scope of this paper.

We also assume that an architecture A_i can be defined in terms of design variables. For instance, in a client-server architectural style, like the one for the systems in Figs. 2 and 3, the design variables might refer to: the number of active devices, the number of service instances deployed on each device, or the average processing time of a service, among others. Each architecture A_i is evaluated using analysis models to compute quality-attribute metrics for the architecture. For instance, a performance model can estimate the latency of client requests, while a reliability model can estimate the failure probability for the system. More generally, an analysis model can be seen as a function over design variables of A_i, plus additional variables (or constants) coming from the system environment (e.g., number of services allowed per device). An architectural configuration, then, is given by an assignment of values to those variables.

Given a set of k objectives to be optimized $OS = \{O_0, O_1, O_2, ..., O_k\}$, let us assume a quality-attribute space that maps each architecture configuration A_i to a multi-valued point in the objective space. Each objective O_j represents a different quality-attribute metric, such as latency, failure probability, or cost. There is a satisfaction threshold for each objective that indicates whether an architecture configuration is a valid (or satisfactory) solution for that objective[2]. For instance, one might specify that a latency objective is met by A_x if the performance analysis for A_x returns a value smaller than a constant $P_{threshold}$ (for a minimization objective). In our example of Fig. 4 $P_{threshold} = 20ms$ and the performance value of A_n is below $P_{threshold}$ after four optimization steps.

A tactic is a transformation that modifies the values of some design variables, while ensuring that the resulting configuration is valid. Architectural tactics are based on standard design mechanisms (or decisions) for improving specific quality attributes [9]. Given an initial architecture A_0, the optimization process generates multiple architectures that are reachable from A_0 by means of tactic sequences (strategies). Suppose we have a strategy given by the sequence $ST_n =< T_1, T_2, T_3, ..., T_n >$ that derives architecture A_n from A_0. Our interpretation problem is how to *answer a why-question for A_n with respect to O_j based on ST_n*. In particular, we propose choosing a subset $RT_n \subset ST_n$ containing the most relevant tactics, along with a description of the tactic effects on O_j. This description might have textual or graphical contents and can be built using predefined templates [10]. Fig. 5 shows a template snippet that explains A_n focused on the performance objective. The template is organized intro three sections: context, rationale and summary. In this case, the template placeholders were instantiated with data from: A_0, A_n, the three objectives and their respective thresholds, the values of the quality-attribute analyses, and the first two tactics (i.e., the relevant ones) from the strategy for A_n.

[2]Other types of optimization, not limited to thresholds, are also possible.

In general, we can then define a ranking function $relevance(A_i, O_j, ST_i) \rightarrow RT_i$ that computes a score for each tactic T_x (in ST_i) with respect to O_j and generates RT_i. We have currently implemented two possible procedures for this function, which are described below.

Instance-based Procedure: This procedure selects those tactics T_x (from ST_i) with the largest effect on the improvement of O_j. Improvement is quantified as the gradient between two consecutive architectures A_i and A_{i+1} in the sequence. The direction of the gradient is also analyzed: for a minimization objective (like response time), the gradient should be negative, while for a maximization objective it should be positive. In our implementation, we considered a gradient to be large enough if it was greater than the average gradient of the sequence. Under this criterion, the first two tactics of A_n (Fig. 4) received the highest scores and were listed in the context section of the template (Fig. 5). If the architect's question were instead about cost, the first and third tactics would have been selected.

Neighborhood-based Procedure: The target strategy ST_i is compared to other similar sequences produced by the optimization so as to discriminate influential tactics. A tactic is said to be influential if its absence (in a sequence) results in a different outcome for O_j. To do so, we codify each sequence in terms of its constitutive tactics, the number of times each tactic was applied, and the values of all the objectives at A_n. The group of tactics being similar to ST_i can be obtained via clustering (e.g. k-means or agglomerative clustering algorithms). In this group, we separate out the tactics that meet O_j (i.e., its threshold) from those that do not, transforming the problem into a binary classification. Then, we train a standard classifier (e.g., a decision tree) and apply a feature importance technique to get a ranking of the most influential tactics for threshold satisfaction. For instance, Fig. 6 lists a ranking of tactics for a neighborhood of T_n under $P_{threshold} = 18$ using permutation feature importance. Note that the results differ from those of the instance-based procedure: `increaseCapacity(device3)` was found to be the main tactic (to be included in the template) and `increaseCapacity(device1)` is not part of the original ST_n. This situation can be attributed to the different information sources (single sequence versus similar sequences) used by the procedures for computing relevance.

IV. Initial Evaluation

To assess the role of tactic-based explanations, as generated with our approach, we conducted an exploratory user study with a group of 12 architects: 8 from academia and 3 from industry. 70% of the participants had more than 10 years of experience and 50% reported having excellent software architecture knowledge. The goal was to gather feedback on the pros and cons of explanations when used by architects in design evaluation tasks. The main instrument was an online questionnaire[3] that presented an architecture reasoning

[3]The questionnaires can be accessed here: https://www.surveymonkey.com/r/HSBXNVJ and https://www.surveymonkey.com/r/HRV68J7

QUESTION (user):
* WHY was **Performance satisfied** with a **responseTime=18ms** in solution **#23**?

ANSWER (tool):
A) Context
* Departing from solution **#0** with initial **responseTime= 52 ms**
* Attempting to satisfy the main **Performance** objective: **responseTime <= 20ms**
* The other qualities had initial values: **totalCost=22$ (Cost)** and **successfulExecution=56% (Reliability)**

B) Rationale
* The following **2** tactics were applied:
  1- **Increase capacity of device2**
    which contributed to **decrease responseTime [31ms]**
      regarding **Cost**:
      * it **negatively affected totalCost [44$] but still below the threshold [75$]**
      regarding **Reliability**:
      * it **further increased successfulExecutionR1 [73%] while above the threshold [50%]**
  2- **Increase capacity of device2**
    which contributed to **decrease responseTime [24 ms]**
      regarding **Cost**:
      * it **negatively affected totalCost [46$] but still below the threshold [75$]**
      regarding **Reliability**:
      * it **kept successfulExecution [73%] while above the threshold [50%]**

C) Summary
Overall, the **main Performance** objective got **improved** (w.r.t. the initial value) and also **satisfied the threshold**;
- but this had a **negative** effect on **Cost**, which got **worse** (w.r.t. the initial value) and **could not satisfy** its threshold;
- and had a **positive** effect on **Reliability**, which got **better** (w.r.t. the initial value) and **also satisfied** its threshold.

Fig. 5. Example of explanation generated using a predefined textual template

Weight	Feature
0.0632 ± 0.0096	t3_increaseCapacityR1_device3
0.0594 ± 0.0175	t1_increaseCapacityR1_device1
0.0575 ± 0.0263	t2_increaseCapacityR1_device2
0 ± 0.0000	t4_decreaseCapacityR1_device1

Fig. 6. Example of key tactics using the neighborhood-based procedure.

exercise to the subjects. The user study was organized as an A/B experiment, in which half of the subjects received the questionnaire without explanations (control group), while the other half received the same questionnaire with an explanation (treatment group). We looked at various quantitative metrics (e.g., time taken to complete each question, correct answers) and also analyzed qualitative feedback.

Starting with an initial architecture A_0 (as in Fig. 2), we ran an optimization process based on the PRISM[4] model checker [11] that explored 2108 intermediate architectures and returned 308 architecture candidates with various tradeoffs among performance, reliability and cost. Then, we picked a random candidate and asked our tool to produce two textual explanations: for performance and reliability. Since we were also interested in alternative explanation formats, we manually created a graphical representation that conveyed the same content as the textual format but with a more visual format (the content and format of both explanation types are exemplified in the online questionnaires). Hence, we ended up with two questionnaire variants: one focused on performance and the other focused on reliability. Each questionnaire variant was divided into three groups: a first was without explanations, a second had textual explanations, and a third had graphical explanations. We used SurveyMonkey to create and distribute the questionnaires to the subjects, and collect their responses.

The base questionnaire contained five design situations and

[4]https://www.prismmodelchecker.org/

questions. Each subject was presented with both the initial architecture and the generated candidate, and was asked about the relationships between the underlying candidate design and the quality-attribute objectives it had to satisfy. Each subject was asked to record the time taken to answer each question. The whole exercise was expected to take 20-30 minutes. The questionnaire also asked about the perceived exercise difficulty, the subject's architecture expertise, her confidence on the answers, and general feedback.

Fig. 7 summarizes the results of the three groups according to the type of explanation provided. We observed that including explanations increased the time needed by the subjects for answering the questions, as expressed in [12]. The extra effort of going through an explanation, however, should allow a subject to gain more confidence in her answers or improve her understanding of the design (e.g., making a larger number of correct deductions). Since architects normally rely on graphical representations for their daily tasks, we conjectured that graphical explanations would be more useful (or take less reading time) than textual explanations. As for the subjects' confidence in their answers, we observed a slight difference in favor of the groups using explanations. However, since most participants were experienced architects, they might have relied on prior experiences to answer confidently.

Analyzing the correctness of subjects' answers (i.e., that they marked the right answers), we found that the groups with and without explanations were equally able to complete the tasks and had the same error rate ($\approx 20\%$). The subjects' seniority could also explain this trend. Interestingly, in the treatment group all mistakes were in cases with textual explanations. These subjects might have spent more time reading through the template than those using a graphical format.

Finally, from the subjects' qualitative feedback, we note that most of them asked for more contextual information in order to analyze the questions properly. For instance, certain tactics could be analyzed differently (or with a higher confidence) if the assumptions behind those decisions were clearer in the explanations. Thus, we think that choosing a subset of key tactics might be not enough, and the explanations should include somehow the *context* around those tactics.

V. Conclusions and Future work

In this work, we presented an approach towards making the results of an architecture optimization tool more comprehensible to human architects. We relied on architectural tactics as shared constructs between the human's mental model (of design decisions) and the optimization mechanisms used by the tool. We also proposed two procedures for reducing the amount of information being presented to the architect to mitigate the cognitive load of the explanations. Along this line, the notion of relevant design decisions (in the form of tactics) was identified. The explanations delivered by our approach are *local* as they expose the decisions for a given architecture rather than the variables of the whole design space.

The evaluation results, although preliminary, show that both the explanation format (e.g., text, charts, layout) and the *context* in which the key decisions are presented are important for architects, and thus, they should be further investigated. As future work, we will explore explainability techniques to deal with different quality-attribute questions for architectural designs. Finally, we will also experiment with other optimization tools and larger case-studies.

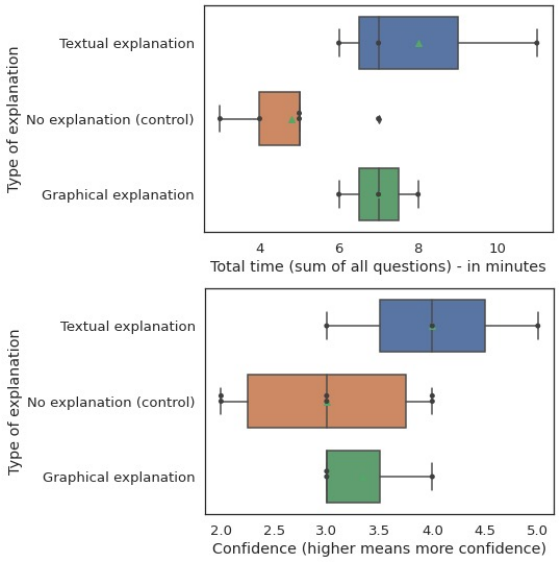

Fig. 7. Time taken and perceived confidence according to subject group.

References

[1] A. Aleti, B. Buhnova, L. Grunske, A. Koziolek, and I. Meedeniya, "Software architecture optimization methods: A systematic literature review," *IEEE Trans. on Soft. Eng.*, vol. 39, no. 5, pp. 658–683, 2013.

[2] A. R. Quesada, J. R. Romero, and S. Ventura, "Interactive multi-objective evolutionary optimization of software architectures," *Inf. Sci.*, vol. 463-464, pp. 92–109, 2018.

[3] S. Procter and L. Wrage, "Guided architecture trade space exploration: Fusing model based engineering amp; design by shopping," in *2019 ACM/IEEE 22nd Int. Conf. on Model Driven Engineering. Languages and Systems (MODELS)*, 2019, pp. 117–127.

[4] A. Busch, D. Fuchß, and A. Koziolek, "Peropteryx: Automated improvement of software architectures," in *2019 IEEE International Conference on Software Architecture Companion (ICSA-C)*, 2019, pp. 162–165.

[5] R. Capilla, A. Jansen, A. Tang, P. Avgeriou, and M. A. Babar, "10 years of software architecture knowledge management: Practice and future," *Journal of Systems and Software*, vol. 116, pp. 191–205, 2016.

[6] Q. V. Liao, D. Gruen, and S. Miller, "Questioning the AI: Informing design practices for explainable AI user experiences," in *Proc. 2020 CHI Conf. on Human Factors in Comp. Systems*. ACM, Apr. 2020.

[7] F. Doshi-Velez and B. Kim, "Towards a rigorous science of interpretable machine learning," 2017, cite arxiv:1702.08608.

[8] J. A. D. Pace and M. R. Campo, "Exploring alternative software architecture designs: A planning perspective," *IEEE Intell. Syst.*, vol. 23, no. 5, pp. 66–77, 2008.

[9] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, ser. SEI series in software engineering. Addison-Wesley, 2003.

[10] R. Sukkerd, R. Simmons, and D. Garlan, "Tradeoff-focused contrastive explanation for mdp planning," in *Proc. 29th IEEE Int. Conf. on Robot Human Interactive Communication*, Virtual, September 2020.

[11] J. Cámara, D. Garlan, and B. Schmerl, "Synthesizing tradeoff spaces of quantitative guarantees for families of software systems," *Journal of Systems and Software*, vol. 152, pp. 33–49, June 2019.

[12] N. Li, J. Cámara, D. Garlan, and B. Schmerl, "Reasoning about when to provide explanation for human-in-the-loop self-adaptive systems," in *Proc. of the 2020 IEEE Conference on Autonomic Computing and Self-organizing Systems (ACSOS)*, Washington, D.C., 19-23 August 2020.