Supporting the Exploration of Quality Attribute Tradeoffs in Large Design Spaces

J. Andres Diaz-Pace[1] , Rebekka Wohlrab[2] , and David Garlan[3]

[1] ISISTAN Research Institute, CONICET and UNICEN University,
Tandil, Buenos Aires, Argentina
[2] Department of Computer Science and Engineering
Chalmers | University of Gothenburg, Sweden
[3] Software and Societal Systems Department, Carnegie Mellon University
Pittsburgh, PA, USA
{andres.diazpace@isistan.unicen.edu.ar, wohlrab@chalmers.se,
garlan@cs.cmu.edu

Abstract. When designing and evolving software architectures, architects need to consider large design spaces of architectural decisions. These decisions tend to impact the quality attributes of the system, such as performance, security, or reliability. Relevant quality attributes might influence each other and usually need to be traded off by architects. When exploring a design space, it is often challenging for architects to understand what tradeoffs exist and how they are connected to architectural decisions. This is particularly problematic in architectural spaces generated by automated optimization tools, as the underlying tradeoffs behind the decisions that they make are unclear. This paper presents an approach to explore quality-attribute tradeoffs via clustering and visualization techniques. The approach allows architects to get an overview of the main tradeoffs and their links to architectural configurations. We evaluated the approach in a think-aloud study with 9 participants from academia and industry. Our findings show that the proposed techniques can be useful in understanding feasible tradeoffs and architectural changes affecting those tradeoffs in large design spaces.

Keywords: Quality attribute tradeoffs · Architecture exploration.

1 Introduction

Designing and evolving a software architecture is usually a challenging activity, as architects need to consider large design spaces that arise from alternative architectural decisions [4, 7]. These decisions tend to have an impact on quality attributes of the system (e.g., performance, security, or reliability requirements), which might interact with each other. For example, achieving excellent performance and reliability at a low cost and with high security is often unfeasible. In practice, stakeholders make tradeoffs between the quality attributes of interest, which in turn affect the decisions that the architect can choose for the system.

This architectural decision problem can be seen as one of multi-objective optimization [1, 15], in which making a set of decisions (and avoiding others)

produces an architectural configuration that fulfills a set of quality attribute goals to varying degrees. However, in large design spaces, it is often difficult for humans to assess how good a set of architectural decisions is for the relevant quality attributes and what tradeoffs those decisions entail. To assist architects, several tools for automated architecture generation and optimization have been proposed [1, 3, 9, 16, 17], which can search through a wide range of configurations and recommend the most promising ones (e.g., those closer to the Pareto front for the relevant quality measures). An architectural configuration is shaped by the decisions being selected (e.g., deploying a service on a device, or inserting an intermediary between components). An architect normally takes a given configuration as the starting point and then relies on an optimization tool for generating and assessing alternative configurations. In some cases, different configurations might be connected to similar tradeoffs. Conversely, in other cases, small variations in a configuration might lead to different tradeoffs. However, in existing tools, the reasons why a generated configuration fulfills a set of quality attributes are normally opaque to architects. This limitation negatively affects their interactions with the tool and the exploration of architectural alternatives.

The concerns above call for forms of explaining a design space [20] to humans, which are currently not supported in optimization tools. This paper presents an approach to explain tradeoff spaces using clustering and visualization techniques. Clustering is used to find groups of configurations sharing similar quality-attribute characteristics (i.e., making similar tradeoffs). Furthermore, configurations are linked to each other based on a distance measure that considers the decisions (or architectural changes) to transition from one configuration to a neighborhood of alternatives. Along with this process, different charts are used to visualize prototypical configurations within a particular group, or to identify differences between groups. The proposed techniques enable us to summarize a large amount of information about quality attributes and related decisions. Thus, we help architects to quickly explore a design space and increase their confidence in the solutions generated by a tool.

To evaluate our approach, we performed a think-aloud study with 9 participants from industry and academia. The goal was to assess how the proposed techniques support architects in understanding tradeoffs and associated decisions within a space. Although at initial stages, we found that our techniques can be beneficial in large design spaces, allowing users to identify key tradeoffs, reason about architectural changes affecting them, and explore related alternatives.

2   Example: A Client-Server Design Space

For illustration purposes, let's assume a simple client-server architecture as schematized in Fig. 1 (left). The relevant quality attributes are performance, reliability, and cost. Client requests go through a load balancer that assigns them to service instances (on the server side) for processing. Service instances can be deployed on two physical devices. Each device is assumed to have a capacity to host up to six service instances. The two devices differ in their hardware characteristics: $device1$ is a low-end device while $device2$ is a high-end device, which

affects their processing and reliability capabilities. In this setting, the processing time for incoming requests depends on the number of service instances, regardless of their deployment. For reliability, the probability of successfully executing a request is maximized when both devices are used or decreases otherwise.
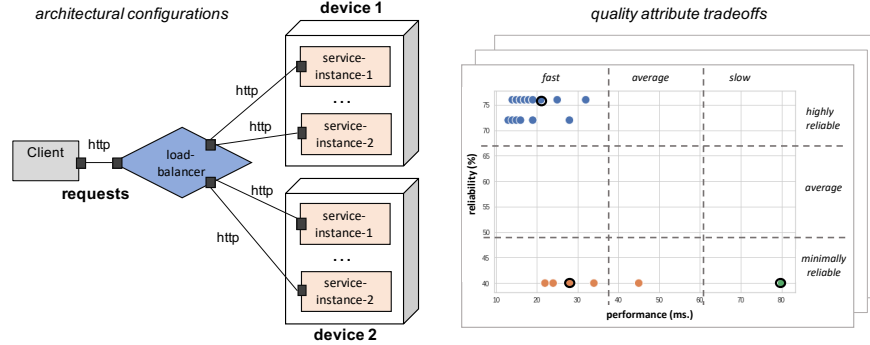
Fig. 1. Simple client-server style and associated tradeoffs.

To quantify the levels of performance, reliability and cost, let's assume that predefined analysis models are provided [15]. These models rely on both the structure and additional properties of a configuration (e.g., device cost, service failure probability, etc.) to compute a variety of measures. For instance, a cost model that sums up the individual costs of all the active services and devices might estimate a total cost of $70 for a configuration. Analogously, models for the processing time and failure probability of a configuration are devised.

In an architectural configuration, the decisions refer to the specific number of service instances and devices being active. As a result, this space has a total of 48 possible configurations with different tradeoffs. For example, the chart of Fig. 1 (right) shows that configurations with fast response times for processing the requests might have either low or high reliability. However, not all combinations of performance and reliability are feasible in this space. In general, it is not always obvious for the architect to determine what tradeoffs a given configuration is associated with and how the quality measures are correlated with each other.

3   Requirements for Tradeoff Explainability

For explaining tradeoff spaces, it is crucial to understand the information needs that stakeholders have. In particular, we focus on the questions that architects might pose when trying to reason about architectural configurations, design decisions, and quality-attribute tradeoffs. First, to select an appropriate configuration, it is important to get an overview of the kinds of tradeoffs that exist in a design space. Afterwards, the corresponding configurations can be assessed individually, along with alternatives having similar or different tradeoffs. In this work, we consider the following questions:

- **Q1**: What are the categories of feasible tradeoffs in a given design space?
- **Q2**: What architectural configurations are representative of each category of tradeoffs? What are the key decisions behind those configurations?

- **Q3**: For a given configuration, which alternatives lead to similar tradeoffs?
- **Q4**: For a given configuration, which alternatives lead to different tradeoffs?

We argue that, by answering these questions, tradeoffs can be understood and the design space can be quickly explored to arrive at one or more configurations that make appropriate tradeoffs. The requirements of a tradeoff explainability approach are thus to: answer Q1–Q4, provide a sufficient level of usability, and be understandable to stakeholders with basic architecture knowledge.

4   Approach

We propose a framework that involves three stages, as outlined in Fig. 2. The output of the first two stages is a repository that enables the creation of a dashboard with visualizations (called explanation charts) for the architect.
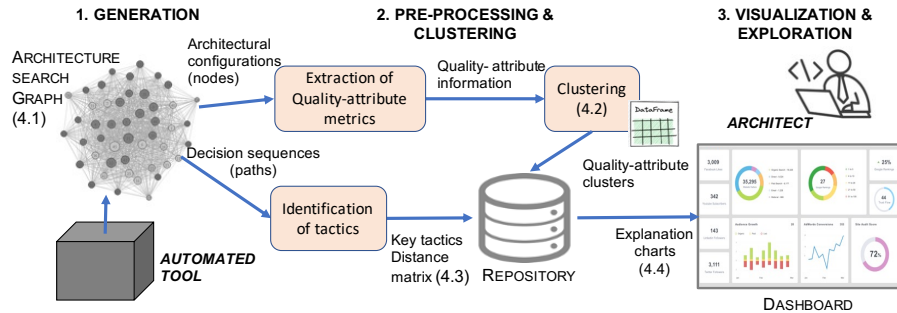
Fig. 2. Stages of the proposed framework.

First, there is an *exploration stage*, in which a (large) number of architectures is generated (or sampled) by an automated optimization tool. This stage must be executed beforehand. Each architectural instance comprises both the architectural configuration and its quality-attribute values. This information is represented by a search graph, which connects an initial architectural configuration to the various configurations that can be derived from it by applying predefined decisions. Next, during the *pre-processing and clustering stage*, the information from the architectural instances is split into two parts. The quality-attribute values are grouped to derive clusters. To do this, a combination of discretization (of numerical values) and clustering techniques is used. The sequences of decisions (paths in the graph) leading to each configuration are identified, and the important decisions for the tradeoffs are then determined. Furthermore, decisions serve to compute a distance metric between sequences. All the artifacts are stored in a repository. Finally, the *visualization stage* provides a set of explanation charts for the architect to get insights on the tradeoff space.

4.1   Design representation using a search graph

We consider a multi-objective architecture optimization [4], and assume an *architectural space* for a family of systems that encompasses all possible architectural

configurations in terms of a (finite) set of *design decisions*. More formally, let $DS = \{A_0, A_1, A_2, ..., A_n\}$ be a design space with n architectural configurations in which each A_i corresponds to a (valid) configuration that results from a sequence of predefined decisions $A_i = < d_{1i}, d_{2i}, ..., d_{mi} >$. Each $d_{ij} = 1$ if the decision was made (for configuration A_i) or 0 otherwise.

In this work, we restrict the possible decisions to *architectural tactics* [5]. An architectural tactic is a design transformation that affects parts of an architectural configuration to improve a particular quality attribute. In our example, a tactic is to deploy a service instance on a given device in order to increase the system performance. Note that the same tactic might have an effect on other quality attributes (e.g., cost). From this perspective, the configurations in DS are linked to one another through the application of tactics. DS can be visualized as a directed graph in which each node represents a configuration, while an edge between two nodes A_i and A_j captures a tactic leading from the former to the latter. A distinctive node A_0 refers to the initial configuration.

In general, an automated tool will be responsible for exploring the design space and generating a (large) graph of configurations. The techniques for populating DS might include specific architectural tools [1], evolutionary algorithms [3, 6], or model checkers such as Prism [7, 14], among others. Since enumerating all the configurations available in the design space is usually computationally unfeasible, only a subset of those configurations will be generated. Our framework does not depend on the tool or the specific search technique, as long as it can expose the decisions being applied for each configuration.

We require a configuration A_i to be assessed with respect to multiple quality attributes (objectives) by means of quantitative measures [15]. Let $QAS = < O_1, O_2, ..., O_k >$ be a *quality-attribute space* with k objectives in which each O_k represents a quality metric (e.g., latency, failure probability, or cost) associated with some architectural configuration. That is, an evaluation function $f : DS \rightarrow QAS$ maps a configuration to a multi-valued vector in $\Re^k$.

4.2 Clustering of the quality-attribute space

To understand the possible tradeoffs, architects need a succinct representation of the quality-attribute space. A suitable technique for this purpose is *clustering* [22], which is a Machine Learning technique for grouping a set of instances in such a way that instances in the same group (or cluster) are more similar to each other than to those in other groups. To do so, a similarity criterion needs to be established. In our case, an instance refers to the quality-attribute vector for a configuration A_i. For the similarity criterion, we rely on the Euclidean distance between vectors (although other metrics could be used).

We are interested in cohesive clusters that capture the main tradeoffs available in the space. To this end, classical algorithms such as *k-means* or *agglomerative clustering* [22] can be applied. In both algorithms, the number of desired clusters is specified beforehand, and the quality of the resulting clusters is assessed with metrics such as the silhouette coefficient [22]. This coefficient assesses the (average) cohesion and separation of a set of clusters, by measuring how similar instances are for their own cluster compared to the other clusters.

Once clusters are identified, we assign a label to each cluster that reflects the quality attributes being traded off in a way that humans can more easily understand. Specifically, we select the cluster *centroid*, which is computed as the mean of the vectors belonging to the cluster. Examples of three clusters were shown in Fig. 1 (right) with their centroids marked. Since a centroid is a numeric vector, we transform it into a label by means of a discretization procedure which partitions the range of values of each quality attribute into an ordinal (or Likert-like) scale. In Fig. 1, the partitioning is indicated by the dotted lines that map to levels of satisfaction (e.g., *fast*, *average*, or *slow* for performance). The cluster label results from the concatenation of the quality-attribute levels of the cluster centroid. In our example using a 3-point scale, the orange cluster gets labeled as *fast/minimally-reliable*, the blue cluster gets *fast/highly-reliable*, and the green cluster gets *slow/minimally-reliable*. Note that after the clustering and discretization, the quality-attribute space is reduced to three groups of tradeoffs. The number of architectural configurations belonging to each cluster might vary, depending on the design space being considered.

4.3   Distance between architectural configurations

An architectural configuration in the search graph is the result of applying specific tactics to the initial architecture. The optimization tool progressively applies different tactics to derive alternative configurations. Thus, a configuration A_i can be represented by a *sequence of tactics* $S_i = < t_{1i}, t_{2i}, ..., t_{mi} >$, which comes from the shortest path between the initial node and a particular node (with a valid configuration) in the search graph. Coming back to our client-server example, we can consider a tactic *increaseCapacity*(?*device*) that deploys (and activates) a new service on a given device[4]. Instantiations of this tactic for specific devices (*device*1, *device*2) will then label the edges of the graph. Note that we consider the sequences as shorter paths in the graph because all the tactics in the example involve "atomic" configuration changes. If more complex tactics were available, other criteria for determining the paths to each alternative should be considered.

By representing configurations as sequences of decisions, we can assess the distance between two configurations in terms of their delta of changes [3]. Given a pair of configurations A_i and A_i, which are derived from sequences S_i and S_j respectively, we model their distance as a function of the differences in the tactics made for S_i and S_j. Our approach currently uses a version of the *hamming* distance[5] [10], although other metrics could be employed. A distance matrix for all tactic sequences is computed in the *pre-processing stage* (Fig. 2).

When the architect wants to explore alternatives for a configuration A_x, all configurations are sorted based on their distance to A_x, and the top-k results with the shortest distance are returned. Upon the architect's request, filters can be applied to select: only configurations within a particular cluster, configurations

---

[4] A tactic *decreaseCapacity*(?*device*) that reverses the effects of adding a service to a device can be also considered in the graph.

[5] In case of sequences of different lengths, we pad the shorter sequence with a special *noOp*() tactic that makes no changes to the architectural configurations

belonging to all clusters, or configurations that exclude a predefined cluster. For instance, Fig. 3 shows the distances between three configurations resulting from two clusters from our example. Configurations A and C are assumed to share the same cluster, while configuration B belongs to a different cluster. The labels associated with each configuration refer to its quality-attribute characteristics (which might slightly differ from those of the cluster centroid). For example, as shown in Fig. 3, one could sacrifice response time (e.g., due to cost concerns) by moving from A towards C, which both decreases the number of active services and uses a cheaper device for them. Alternatively, one could make changes to A so as to reach B, which offers both higher reliability and smaller response time, because services are deployed in two devices. Alternative C causes a slight variation in the tradeoff of A for performance and reliability, while alternative B leads to a better tradeoff for both qualities. In general, intermediate configurations might need to be traversed (in the graph) to move between two particular alternatives. Note also that the distances refer to changes in the design space, rather than to cluster differences in the quality-attribute space.
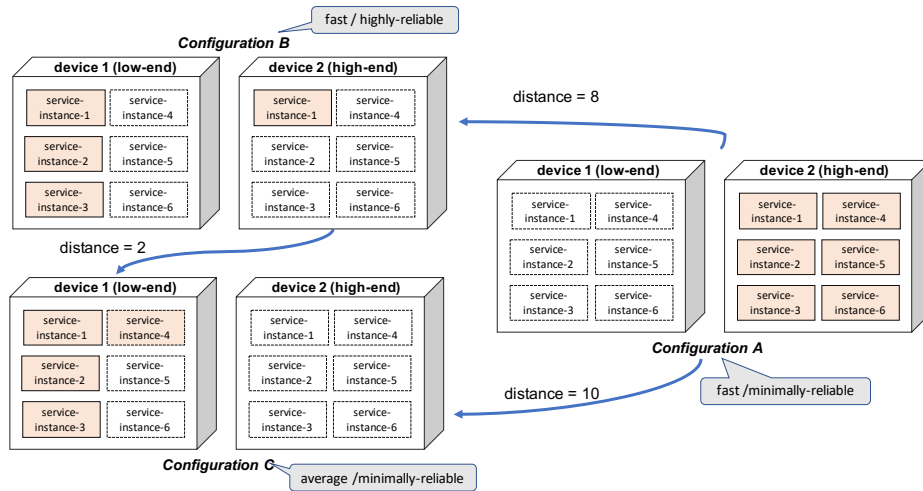

Fig. 3. Alternative configurations and tradeoffs based on decision changes.

4.4 Explanation charts

In the *visualization stage*, the architect goes through a series of charts that shed light on different aspects of the design space. This exploratory process is structured as a suggested workflow of activities. Each activity involves a specific chart targeted to answer questions Q1-Q4. The four available charts are illustrated in Fig. 4. The suggested order for an architect to use them is clockwise, as indicated by the numbers in the figure. The charts were designed and adjusted iteratively by the authors, according to the notion of *pretotypes* [19], which allows one to test ideas at a low cost before building a (prototype) tool.

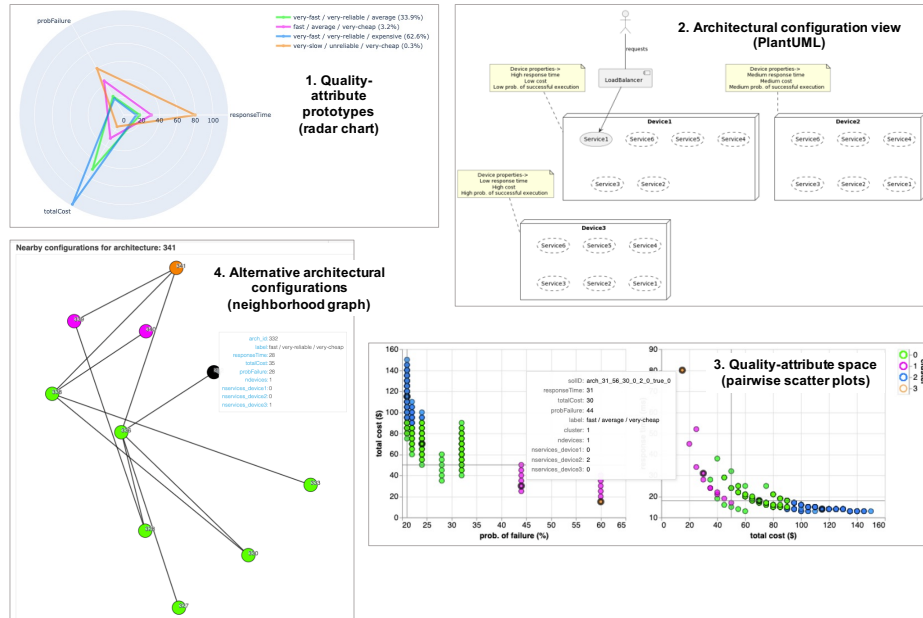   We briefly describe below the main characteristics of each chart type:

Fig. 4. Types of explanation charts.

1. **Quality-attribute prototypes**. This radar chart displays the values of the cluster centroids with respect to the quality attribute goals. As the initial view, it presents the main tradeoffs of the quality-attribute space to the architect, using the centroids as representative instances (or prototypes) of the feasible tradeoffs. This chart is intended to address **Q1**. The chart legend additionally shows the labels and the percentage of architectural configurations (from the whole space) per cluster.

2. **Architectural configuration view**. To focus on a given prototype or cluster (from chart #1), the architect can inspect one or more architectural configurations related to that cluster. This is motivated by the need to drill down into the structure of configuration and its underlying decisions. The chart targets **Q2** and enables an understanding of the design space. Here, we use PlantUML for our client-server example, although the notation is dependent on the architectural models being captured by the space.

3. **Quality-attribute space.** This chart gives a detailed view of all the architectural configurations, their contributions to the different quality-attribute measures, and how configurations are grouped into clusters. It complements chart #1 by showing all possible tradeoffs, in order to address **Q1** and **Q2**. This chart is interactive, allowing architects to select specific points and display basic information about the configurations or cluster labels.

4. **Alternative architectural configurations**. After the insights exposed by the previous charts, the architect might want to understand how to move from a given configuration to another with different quality-attribute char-

acteristics. This chart creates an interactive graph that takes a target configuration and connects it to a set of nearby configurations that result from making "small" changes to the decisions for the target configuration. The alternative configurations might belong either to the same cluster as the target or to different clusters. This graph chart seeks to address **Q3** and **Q4**. The target can be any configuration from charts #1 or #3. The construction of the graph is based on the architectural distance described in Section 4.3.

5   Study Design

To evaluate the effectiveness of our explanation framework, we applied it to an extended version of the client-server problem presented in Section 2 in two ways: (i) exploring and assessing configuration variants and tradeoffs using a pre-generated space, and (ii) conducting a think-aloud study to evaluate our findings and the role of the explanation charts[6].

5.1   Client-server design space

We set a client-server style with up to three available servers (devices), each with a capacity to deploy a maximum of six services. To model tradeoffs between performance (latency), reliability (probability of failure), and cost (total deployment expenditure), we assigned different characteristics to each device, in terms of high-end hardware (i.e., very good processing capabilities, low failure rate, and high cost), mid-range hardware, and low-end hardware (i.e., minimum processing capabilities, higher failure rate, and low cost). The decisions in this space include: (i) adding one (active) service to any device, or (ii) removing an already-deployed service from a device. The choice of the device depends on its hardware characteristics.

We departed from an initial configuration with no services allocated to devices, and ran an optimization procedure based on the Prism model checker [14]. This way, we generated a large number of architectural configurations for our system to evaluate the feasibility of the proposed framework. In principle, other search strategies (e.g,, evolutionary algorithms) could have been used, as the approach is mostly independent of how the optimization part is implemented.

5.2   User study

The think-aloud method is a technique to investigate problem-solving processes and participants' cognitive models [12], in which participants vocalize their thought process while working on a task. We chose a think-aloud study because it can provide insights into how the clustering and visualization techniques proposed in our framework can facilitate the participants' architectural reasoning.

The study consisted of a series of design sessions with 9 participants from both academia and industry. The selected participants had various roles in their organizations (e.g., university faculty, researcher, PhD student, industry practitioner, and senior engineer) and varying degrees of architectural knowledge and

---

[6] Notebook: `https://shorturl.at/jyCX3` - Tasks: `https://shorturl.at/lHUO8`

experience (from 2 to more than 20 years in the architecture field). All sessions were recorded and had a duration of 45 minutes approximately.

A session involved four phases: introduction, learning, testing, and post-mortem. The session started with an introduction to the architectural problem, and was facilitated by one of the authors. As mentioned above, this problem was based on the design space already investigated (Section 5.1). After the introduction, there was a learning phase in which the participant assumed the role of an architect and was asked to explore the space of quality-attribute trade-offs and candidate configurations by means of the explanation charts. For this phase, we provided a *Jupyter* notebook in *Google Colab* that included prede-fined Python functions for a user to load the space as a dataset and generate the charts. Furthermore, some Python functions admit parameters to adjust the chart behavior (e.g., the name of the architectural configuration to be inspected, or the number of alternatives to show in the graph). We decided not to include the quality-attribute space chart (pairwise scatter plots) in this study, since it provides detailed information about the clusters but can be complex to grasp for unfamiliar users. To avoid long sessions, we considered it could be substituted by the radar and graph charts in the interviews, without losing much information.

The effectiveness of the charts was assessed during the testing phase, in which a set of glitch detection tasks and prediction tasks was presented to the partic-ipant [11]. In the glitch detection tasks, the subjects identify things that are wrong in a system or explanation; while in the prediction tasks, the subjects have to predict the results of certain design decisions and explain their predic-tions. Both kinds of tasks referred to design situations based on the presented charts. These tasks enabled us to analyze the participant's reasoning process and whether it was aided by the mechanisms of the explanation charts.

At the end, we conducted a short questionnaire to measure the satisfaction levels of the participants when using the charts. To do this, we used a list of Likert-scale questions. Apart from satisfaction, we also focused on the partici-pants' expectations as well as on areas of improvement for the framework.

6   Findings

For our client-server example, Prism returned a graph with a design space of 342 configurations. Each configuration was evaluated with (simplified) analytical models to estimate values of cost, latency, and probability of failure. A visual inspection of the resulting tradeoff space showed that it had enough diversity and coverage. Fig. 5 depicts both the design and quality-attribute spaces. Each path in the graph is a sequence of design decisions. The sequence length to move between configurations ranges from 1 to 18 decisions.

In the clustering process, we sought a balance between having a small number of clusters and achieving a high silhouette coefficient. We applied hierarchical clustering with four clusters, which yielded a silhouette coefficient of 0.52. The obtained clusters have unique combined labels, which are derived from a dis-cretization of the ranges for each quality attribute into five bins. In principle,
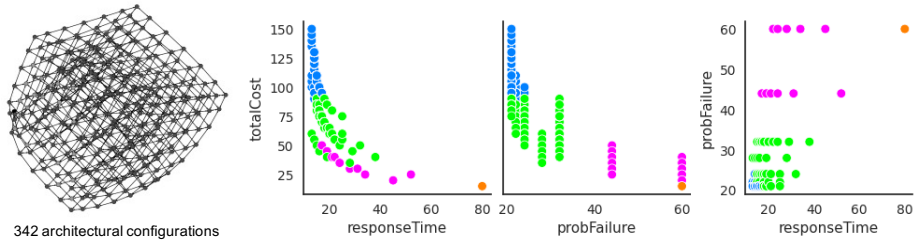
Fig. 5. Design space (graph) and quality-attribute space (pairwise scatter plots) for the client-server space. Colors refer to the clusters exposing four tradeoffs.

this discretization enables 125 possible tradeoffs, but only 19 of them were reachable in the space. Thus, our clustering reduced the tradeoff space in $\approx 80\%$.

For the user study, we categorized the findings according to the chart types, focusing on the understandability of the charts (and the design space thereof) as well as on participants' satisfaction. An overview of the questions and answers from the post-mortem questionnaire is given in Fig. 6. The participants indicated that they were generally satisfied with the information conveyed by the charts. During the learning phase, all participants were able to understand the purpose of the three charts, and to use them for architectural reasoning in the testing phase. Most glitches were detected by the participants. The most difficult glitches and prediction tasks were those related to the graph chart, particularly for less experienced participants.
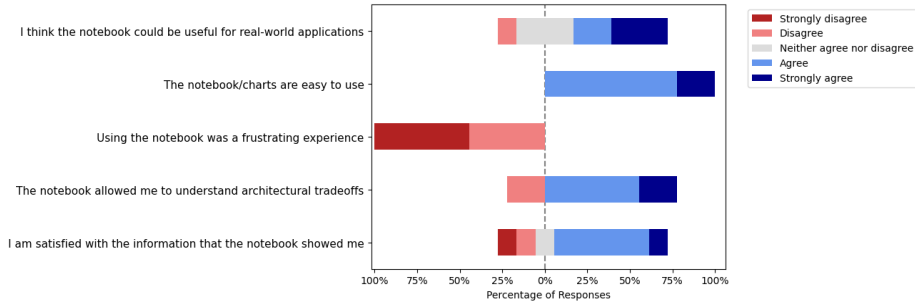
Fig. 6. Likert-scale answers of participants of the user study.

The radar chart was judged as the easiest to use by most participants. During the testing phase, it was perceived as straightforward to apply. This might be due to the fact that architects are often exposed to similar charts. In fact, other visualization tools [9] have proposed radar charts for tradeoffs. A PhD student stated: *"The radar chart with the clusters shows you quite easily what types of solutions are the most likely ones to satisfy your needs"*. As for the labels, they were intuitive for the participants, although some asked questions about the value ranges for the labels, and moreover, about the quality-attribute thresholds that should be considered in order to weight each cluster prototype. On the downside, some participants argued that despite the chart provides an overview of the tradeoffs, it did not seem actionable with respect to decision-making.

The architectural configuration chart was used by participants to drill down into the architectural structure of a particular prototype. Participants usually did not explore more than one configuration with the chart. During the testing phase, the annotations on the devices were a key element for detecting glitches. This might suggest that enriching architectural views with quality-attribute annotations can help design reasoning. However, this aspect likely depends on the size and architectural notation used for the views. Along this line, one researcher noted: *"If I had a system with dozens of components, where there is only a slight difference between configuration A and configuration B when I'm looking at the alternatives, it'll take me much longer to understand what the differences are between these alternatives, and I'd have to trust more the tool to do the job right ... I might need something that shows me the differences between the architectures"*.

When it comes to the graph chart, we observed mixed impressions. A number of participants found the chart difficult to interpret and use later in the glitch detection and prediction tasks. This situation was evident for those participants with less architecture experience. For instance, the implications of the decisions made when transitioning from one configuration to another were not easy to be reasoned about. Despite this complexity, other participants argued that it was useful to explore alternatives by means of small configuration changes. One practitioner said: *"The graph was probably the most challenging to use but also potentially getting to be the most powerful"*. Another participant stated: *"[The graph chart] shows me a what-if analysis ... it lets me do an analysis of different tactics and look at what is their impact in terms of the tradeoffs I would get"*. We collected suggestions to enhance this chart, such as: clarifying the tactic names in the edges, linking the nodes to architectural views (configurations), and (again) adding quality-attribute thresholds. During the learning phase, three participants raised questions about the effort or complexity (e.g., development cost, or deployment cost) of applying the decisions shown in the graph.

Overall, we can conclude that the charts were effective in helping to expose the tradeoffs of the space, and to a lesser extent, in helping the participants reason about decisions affecting those tradeoffs. Having a better focus on the decisions that architects could make is one of the areas of improvement that we identified from the user study. In retrospect, our findings indicate that the information shown in the notebook provides insights about the tradeoffs and configurations, but it might lack contextual information about the problem that the architect is trying to solve. In this regard, one practitioner said during a session: *"I'm trying to arrive at a decision, not just see that there are tradeoffs"*. Furthermore, two practitioners mentioned that the charts should be integrated with tools that architects and developers use in their daily work (e.g., infrastructure as code, dashboards, or IDEs) for assessing options and making decisions.

6.1   Threats to validity

Internal validity. The results of the clustering process depend on both the characteristics of the quality-attribute space and the (hyper-)parameters used for the algorithm (e.g., choosing the number of clusters). The cluster boundaries might not be always clean. Furthermore, using the cluster centroid (and

its associated label) to characterize the tradeoffs of all the cluster members is an approximation. Not all the instances belonging to a cluster might have the same tradeoff posed by the centroid, and thus slight tradeoff variations might appear in the space. In the client-server example, applying other clustering algorithms (e.g., k-medoids) could produce different results. All these factors can affect the participants' interpretation of the radar chart and parts of the graph chart. The graph chart relies on both the tactic sequences for each configuration and a distance criterion for sequences. The sequences can be seen as proxies for representing the configurations, but they omit some architectural characteristics. Thus, the sequences were a good representation for our client-server example, but they might fall short for dealing with other architectural styles. Regarding the architectural distance, we implemented a hamming distance under the assumption that all the tactics have the same weight (or involve a similar amount of changes). This might not hold if a different set of tactics is considered.

Construct validity. When talking about concepts like "tradeoffs" or "software architecture", our participants might have different interpretations as to what they mean. Especially due to their abstract nature, these constructs can be difficult to understand. To mitigate the threat, we discussed the example scenario and concepts at the beginning of the interviews. We asked our participants to describe the scenario in their own words to understand their ways of thinking about tradeoff-related issues. In case they were unsure, we gave explanations of the key constructs and ensured that our views were aligned.

Conclusion validity. While we did not aim to arrive at statistically significant results, conclusion validity is still relevant for our study. The reliability might have been compromised by having a sample of nine participants with limited time to work with the notebook during the sessions. Collecting data from a larger number of participants would have led to more information and richer feedback about the framework. To mitigate this, we aim to be transparent about our research method and study materials. We thoroughly discussed and refined the materials to avoid issues such as potentially incoherent structure, overly complex visualizations, or poor wording in tasks and questions.

External validity. The automated generation of a large graph of alternatives, which also includes the information required by our framework (e.g., decisions applied at each step, quality-attribute values for each alternative), can be challenging and might not feasible (or accurate) for any system or optimization algorithm. This aspect might prevent the exploitation of the techniques presented in the paper. Furthermore, our study does not have a broad generalizability, as it was exercised on a small architectural problem. The goal was to present a think-aloud study focusing on the practical usage of the explainability charts by humans. We selected the participants trying to achieve a coverage of different profiles. Involving different participants helped us get a variety of perspectives on the topic and strengthen external validity. Another threat is the presence of the authors, who assumed a central role in developing the notebook and facilitating the sessions. These factors might lead to the participants responding more positively. As mitigation, we stressed that they should openly share their

thoughts and that improvement suggestions were welcome. Our results indicate that the participants followed these instructions and shared criticisms.

7   Related Work

Several tools for automated architecture optimization that generate a set of alternatives have been proposed [1,3]. These tools work mostly as black boxes, and their internal search space is not comprehensible by humans. As a result, architects might not trust the proposed architecture candidates. Recent approaches, like *SQuAT-Viz* [9] and *Voyager* [16], have investigated visualization techniques for helping architects to understand tradeoffs, and have also evaluated their usability. Among other techniques, *SQuAT-Viz* [9] uses radar charts and scatter plots for the quality-attribute space, although they show all possible combinations of tradeoffs. *Voyager* [16], in turn, combines tradeoff analysis with architectural structure visualizations alongside, highlighting the need to connect these two spaces, which is a shared concern with our framework. *Voyager* does not consider strategies for reducing the size of the quality-attribute space (e.g., via clustering) or navigating related configurations (e.g., like our graph chart).

Other authors have attempted to explain tradeoff spaces using dimensionality reduction and clustering techniques. For instance, Camara et al. [8] propose PCA (Principal Component Analysis) loading plots to relate quality-attribute and architectural variables. In the robot planning domain [20], contrastive explanations of tradeoffs have been developed. This kind of explanations compares a selected policy to Pareto-optimal alternative plans and describes their quality-attribute impact on the user. Also in the planning domain, Wohlrab et al. [21] complement the PCA plots of Camara et al. [8] with clustering and decision trees. The usage of the clusters differs from our framework, as they refer to policies sharing similar characteristics and provide a high-level tradeoff explanation. The clustering process is applied on top of the loading plots, which often implies some information loss when transforming the space to a 2D representation. Furthermore, clusters are explained using bar charts showing feature means, which might not be an intuitive visualization for humans. These works center on information reduction techniques for the design space, but user studies about their effectiveness have not been reported yet.

The *GATSE* tool [17] allows architects to visually inspect *AADL* (Architecture Analysis and Description Language) models from a previously computed dataset. It offers several visualizations to support quality-attribute analyses of AADL models (e.g., via a Pareto diagram), enabling the architect to focus on regions of the quality-attribute space to narrow down or deepen the search for alternatives. This interaction mechanism is called "design by shopping".

Kinneer and Herzig [13] investigate metrics of dissimilarity and clustering for a set of spacecraft architectures within a space mission domain. Since a large number of architecture candidates are automatically synthesized, but some candidates might be similar to each other, the architect has to waste time sifting through the space. Thus, a clustering process based on PAM (Partitioning Around Medoids) is proposed to group the architectures and select a represen-

tative instance from each group. The clustering is tied to the notion of distance between architectures. Based on user studies that identify correlations between clustering and human judgements, the authors highlight the role of human perception when different stakeholders explore the space.

8   Conclusions

In this paper, we discussed some requirements for improving an architect's understanding of design spaces regarding the interplay between tradeoffs and architectural decisions. To this end, we presented an approach that relies on clustering and visualization techniques. An initial version of our framework was evaluated on an architectural problem with a think-aloud study. This study confirmed our hypothesis that design spaces can be summarized to a handful of quality-tradeoffs and related architectural decisions, and also provided us with feedback to improve the explanation charts and underlying techniques. This suggests more focus on characterizing the architecture space, which has received less attention in the literature in comparison to the quality-attribute space.

As future work, we plan to test our framework for larger design spaces, whether generated by existing optimization tools [3, 17] or by humans [18]. As the spaces grow larger, mechanisms to extract the main paths of decisions and tradeoffs will become increasingly important. We think that the notion of "policy summaries" [2] can be adapted to work with the graphs of alternatives (within the architectural space) to extract a sub-graph of decisions that contribute the most to the quality attributes of interest for the architect or stakeholders.

References

1. Aleti, A., Buhnova, B., Grunske, L., Koziolek, A., Meedeniya, I.: Software architecture optimization methods: A systematic literature review. IEEE Trans. on Soft. Eng. **39**(5), 658–683 (2013)
2. Amir, O., Doshi-Velez, F., Sarne, D.: Summarizing agent strategies. Autonomous Agents and Multi-Agent Systems **33**(5), 628–644 (sep 2019). `https://doi.org/10.1007/s10458-019-09418-w`
3. Arcelli, D., Cortellessa, V., D'Emidio, M., Di Pompeo, D.: Easier: An evolutionary approach for multi-objective software architecture refactoring. In: 2018 IEEE Int. Conf. on Software Architecture (ICSA). pp. 105–10509 (2018). `https://doi.org/10.1109/ICSA.2018.00020`
4. Bachmann, F.: Designing software architectures to achieve quality attribute requirements. IEE Proceedings - Software **152**, 153–165(12) (August 2005)
5. Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice. SEI series in software engineering, Addison-Wesley (2003)
6. Busch, A., Fuchß, D., Koziolek, A.: Peropteryx: Automated improvement of software architectures. In: 2019 IEEE Int. Conf. on Software Architecture Companion (ICSA-C). pp. 162–165 (2019). `https://doi.org/10.1109/ICSA-C.2019.00036`

7. Cámara, J., Garlan, D., Schmerl, B.: Synthesizing tradeoff spaces of quantitative guarantees for families of software systems. Journal of Systems and Software **152**, 33–49 (June 2019)
8. Cámara, J., Silva, M., Garlan, D., Schmerl, B.: Explaining architectural design tradeoff spaces: A machine learning approach. In: Software Architecture: 15th European Conf., ECSA 2021, Sweden, Proceedings. p. 49–65. Springer-Verlag, Berlin, Heidelberg (2021). `https://doi.org/10.1007/978-3-030-86044-8_4`
9. Frank, S., van Hoorn, A.: Squat-vis: Visualization and interaction in software architecture optimization. In: Software Architecture - 14th European Conf., ECSA 2020 Tracks and Workshops, Proceedings. vol. 1269, pp. 107–119. Springer (2020)
10. Hamming, R.W.: Error detecting and error correcting codes. The Bell System Technical Journal **29**(2), 147–160 (1950). `https://doi.org/10.1002/j.1538-7305.1950.tb00463.x`
11. Hoffman, R.R., Mueller, S.T., Klein, G., Litman, J.: Metrics for explainable ai: Challenges and prospects. ArXiv **abs/1812.04608** (2018)
12. Jaspers, M.W., Steen, T., van den Bos, C., Geenen, M.: The think aloud method: a guide to user interface design. International Journal of Medical Informatics **73**(11), 781–795 (2004). `https://doi.org/10.1016/j.ijmedinf.2004.08.003`
13. Kinneer, C., Herzig, S.J.I.: Dissimilarity measures for clustering space mission architectures. In: Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems. p. 392–402. MODELS '18, ACM, New York, NY, USA (2018). `https://doi.org/10.1145/3239372.3239390`
14. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: Proc. 23rd Int. Conf. on Computer Aided Verification (CAV'11). LNCS, vol. 6806, pp. 585–591. Springer (2011)
15. Martens, A., Koziolek, H., Becker, S., Reussner, R.: Automatically improve software architecture models for performance, reliability, and cost using evolutionary algorithms. In: Proceedings of the 1st Joint WOSP/SIPEW Int. Conf. on Performance Engineering. p. 105–116. ACM, New York, NY, USA (2010). `https://doi.org/10.1145/1712605.1712624`
16. Mashinchi, J., Cámara, J.: Voyager: Software architecture trade-off explorer. In: Software Architecture - 14th European Conf., ECSA 2020 Tracks and Workshops, Proceedings. vol. 1269, pp. 55–67. Springer (2020)
17. Procter, S., Wrage, L.: Guided architecture trade space exploration: Fusing model based engineering amp; design by shopping. In: 2019 ACM/IEEE 22nd Int. Conf. on Model Driven Eng., Languages and Systems (MODELS). pp. 117–127 (2019)
18. Sanchez, C.C., Capilla, R., Staron, M.: Estimating the complexity of architectural design decision networks. IEEE Access **8**, 168558–168575 (2020). `https://doi.org/10.1109/ACCESS.2020.3023608`
19. Savoia, A.: The Right It: Why So Many Ideas Fail and How to Make Sure Yours Succeed. Harper-Collins (2019)
20. Sukkerd, R., Simmons, R., Garlan, D.: Tradeoff-focused contrastive explanation for mdp planning. In: Proc. 29th IEEE Int. Conf. on Robot Human Interactive Communication. Virtual (September 2020)
21. Wohlrab, R., Cámara, J., Garlan, D., Schmerl, B.: Explaining quality attribute tradeoffs in automated planning for self-adaptive systems. Journal of Systems and Software **198**, 111538 (2023). `https://doi.org/https://doi.org/10.1016/j.jss.2022.111538`
22. Xu, D., Tian, Y.: A comprehensive survey of clustering algorithms. Annals of Data Science **2**, 165 – 193 (2015)