

The Architect in the Maze: On the Effective Usage of Automated Design Exploration

J. Andres Diaz-Pace 

andres.diazpace@isistan.unicen.edu.ar
ISISTAN, CONICET and UNICEN University
Tandil, Buenos Aires, Argentina

David Garlan 

garlan@cs.cmu.edu
Software and Societal Systems Department
Carnegie Mellon University
Pittsburgh, PA, USA



ABSTRACT

Designing a software architecture that satisfies a set of quality-attribute requirements has traditionally been a challenging activity for human architects, as it involves the exploration and assessment of alternative design decisions. The development of automated optimization tools for the architecture domain has opened new opportunities, because these tools are able to explore a large space of alternatives, and thus extend the architect’s capabilities. In this context, however, architects need to efficiently navigate through a large space and understand the main relations between design decisions and feasible quality-attribute tradeoffs in a maze of possible alternatives. Although Machine Learning (ML) techniques can help to reduce the complexity of the task by sifting through the data generated by the tools, the standard techniques often fall short because they cannot offer architectural insights or relevant answers to the architect’s questions. In this paper, and based on previous experiences, we argue that ML techniques should be adapted to the architecture domain, and propose a conceptual framework towards that goal. Furthermore, we show how the framework can be instantiated by adapting clustering techniques to answer architectural questions regarding a client-server design space.

KEYWORDS

Design exploration, automated tools, applied machine learning, quality attributes, explainability.

ACM Reference Format:

J. Andres Diaz-Pace  and David Garlan . 2024. The Architect in the Maze: On the Effective Usage of Automated Design Exploration. In *2024 International Workshop on Designing Software (Designing '24)*, April 15–14, 2024, Lisbon, Portugal. ACM, New York, NY, USA, 6 pages. <https://doi.org/10.1145/3643660.3643947>

1 INTRODUCTION

Designing a software architecture to meet its main requirements is a complex and frequently error-prone process for human architects. This process usually involves exploring design options, and

assessing and making decisions (e.g., patterns, tactics, or technology choices) to address a set of quality attributes (e.g., performance, reliability, or cost, among others) that trade off with each other. The exploration proceeds in iterations until the architect reaches a solution that best fulfills the architectural drivers [3].

Since the process above is mostly manual, the number of design decisions and alternatives analyzed by humans usually comprises a relatively small set of options, because the complexity of exploring a large design space is beyond humans’ cognitive capabilities. Over the last years, however, several architecture tools relying on automated search and optimization have been developed [1, 12, 13]. These tools are able to search through a wide range of alternatives and recommend the most promising ones (e.g., those close to the Pareto front) for the objectives posed by the architect (e.g., quality-attribute metrics). A first challenge for architecting using these tools is that their working is opaque to the architect. Therefore, it is difficult for a human to understand how and why a given solution was recommended, particularly in cases of multi-objective optimization that imply quality-attribute tradeoffs. In general, the architect’s mindset is driven by abstractions such as quality attributes, tradeoffs, patterns and tactics, which do not match the abstractions internally employed by the tools. This situation creates a gap between the tool outputs and the architect’s expectations.

As an automated tool runs its search for solutions, it generates vast amounts of data. In this context, some approaches have used Machine Learning (ML) techniques (e.g., dimensionality reduction, decision tree learning, and clustering, among others) to process the resulting data and provide digested insights to architects about the design exploration [4, 9, 14]. Nonetheless, such techniques are not primarily designed to address architects’ concerns and their outputs are often not directly usable by architects. For instance, applying a dimensionality reduction technique on a multi-variate dataset and then using clustering might not shed light on the key architecture variables that the architect might act upon, or quality-attribute variables exposing tradeoffs. Thus, a second challenge is how to adapt ML techniques to serve the architect’s information needs. In previous works, we have investigated how ML techniques can be used to answer design questions with a focus on quality-attribute tradeoffs [6], and also performed user studies [7].

These challenges about the opaqueness of automated tools, and the mismatches between the architect’s abstractions and those used by the optimization and ML techniques, negatively affect the architect’s trust in the generated solutions and diminishes the value of an automated design exploration. This calls for architecture-driven explainability mechanisms for design spaces. In this paper, we propose a conceptual framework towards that objective, which

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

Designing '24, April 15–14, 2024, Lisbon, Portugal

© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 979-8-4007-0563-2/24/04

<https://doi.org/10.1145/3643660.3643947>

is outlined in Fig. 1. The main building blocks in this framework are: (i) the identification of architecturally-relevant information needs that can be translated to questions for an automated tool (step ③), and (ii) the provision of mechanisms for combining standard ML techniques and tailoring them to an architectural context (step ⑤).

To demonstrate our approach, we present an example of applying the framework to a client-server design space using clustering. We provide a walkthrough of how clustering can be tailored to condense the spaces of both architecture configurations and quality attributes, and present visualizations helping to address the architect's information needs. The rest of the paper elaborates on the different phases of our framework, discussing pros and cons of the proposed techniques, and closes with perspectives for future work.

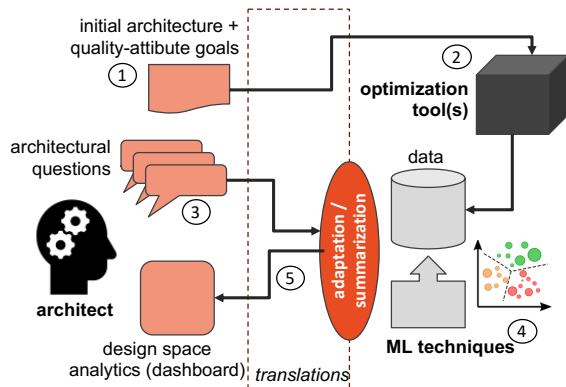


Figure 1: Conceptual framework for effective design exploration by human architects.

2 ARCHITECT'S INFORMATION NEEDS

A conceptualization of a design space typically involves two spaces: (i) the space of architectural configurations (*AC*), and (ii) the quality-attribute (*QA*) space, as depicted in Fig. 2. The *AC* space (or search space) refers to all the feasible architectures that can be derived from a collection of design decisions. Often times, the architect starts with an initial architecture configuration (step ① in Fig. 1) and asks an optimization tool to generate several children configurations by applying predefined transformations (e.g., refactoring operations, tactics, etc.) to the initial architecture (step ②). This search cycle is repeated for each generated architecture in order to derive additional configurations, until a maximum number of iterations is reached or a property is met. The architecture configurations returned by the tool are often arranged as a graph-like structure, in which the nodes correspond to the configurations and the directed edges capture the transformations between a source and a target configuration. The *QA* space refers to the quality-attribute values resulting from evaluating the architecture configurations in the *AC* space. The *QA* space is defined by the attributes of interest for the architect (e.g., performance, reliability, etc.).

In this context, the exploration and understandability challenges for a human stem from the complexity (e.g., in terms of size and number of dimensions) of each individual space, and also from the correspondences (or mapping) between the instances in the two spaces. That is, any configuration in one space is mapped to multi-valued instance on the other space. For this reason, we argue that a human-oriented strategy for dealing with a large design space

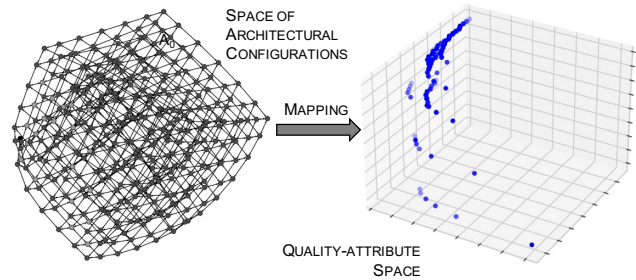


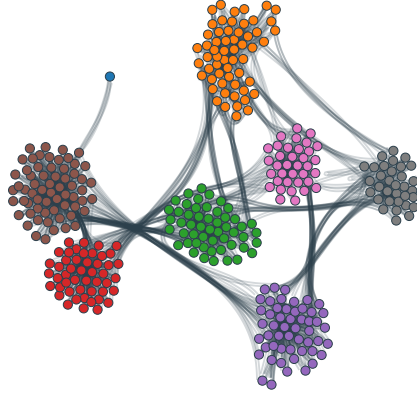
Figure 2: A typical design space: a graph of architecture configurations (left) mapped to a multi-valued space (right).

should be primarily driven by architectural questions that seek to address specific information needs of the architect (step ③). These questions can pertain to the *AC* space, to the *QA* space, or to both of them. Based on prior works and user studies [6, 7], a sample of typical architectural questions are listed below.

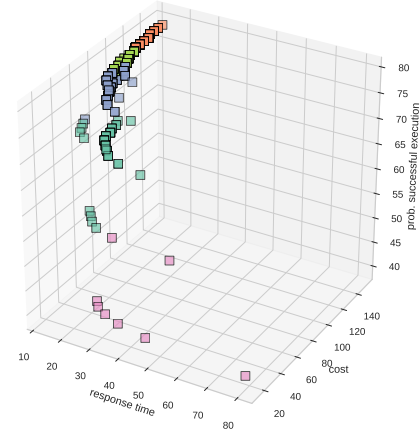
- **Q1:** What are the categories of feasible quality-attribute tradeoffs? (*QA* space)
- **Q2:** What architectural configurations are representative of each category of quality-attribute tradeoffs? (*QA* and *AC* spaces)
- **Q3:** What are the categories of architectural configurations?
- **Q4:** How are the categories of architectural configurations and tradeoffs correlated?
- **Q5:** For a given architecture configuration, which alternative configurations lead to similar quality-attribute tradeoffs? (*AC* and *QA* spaces)
- **Q6:** For a given architecture configuration, which alternative configurations lead to different quality-attribute tradeoffs? (*AC* and *QA* spaces)

We argue that answering these types of questions above requires: (i) summarization mechanisms that can identify representative instances and features in the two spaces; (ii) a translation of those instances and features to architectural abstractions, and (iii) appropriate visualizations to render those abstractions to the architect. ML techniques can support summarization, but if used in a standalone fashion, they often fail to convey abstractions that are architecturally meaningful to architects. In sections 3 and 4 we exemplify how these capabilities can be achieved.

Client-server model problem. Let us consider a design space for a family of systems that adhere to a client-server architectural style [7]. In this style, the client submits requests through a load balancer that assigns the requests to a number of available service instances for processing. A service instance is hosted on a device. In terms of possible architecture configurations, let us assume three devices, each one with the ability to host up to six service instances. Each device might have different hardware characteristics, which influence its cost, processing power, and level of availability. The main quality attributes of the system are: average processing time for individual requests (i.e., latency), deployment cost for the whole system (i.e., services plus devices), and global system availability



(a) Agglomerative clustering of the AC space (7 clusters), incorporating the connectivity of the nodes. Nodes correspond to architecture configurations and edges show design transformations among them.



(b) Clustering of the QA space using k-Medoids (5 clusters). Each point corresponds to a different quality-attribute value resulting from the feasible architecture configurations.

Figure 3: Basic clustering of the AC and QA spaces (partial support for Q1, Q2 and Q3).

(i.e., probability of having at least one device serving requests). This design space entails 342 possible architecture configurations with different values for performance, cost and availability. The configurations were generated with the PRISM model checker [10] (step ② of the framework), although other tools could have been used for this purpose. The generated data is shown in Fig. 2.

For simplicity, an architecture configuration in this case can be defined by the number of active services per device. Some examples of architecture configurations are given in Fig. 6, which also shows the hardware characteristics of each type of device. More complex configurations, including variations in the properties of individual services and devices, are possible but left out of scope for this paper.

3 APPLYING ML TECHNIQUES

A first, intuitive technique to tame the complexity of our example design space is *clustering* [15], which corresponds to step ④ in Fig. 1. Clustering is a type of unsupervised learning 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. An instance is a vector of numeric values, and similarity among instances is computed by means of a distance function (e.g., Euclidean or Cosine). In the QA space, instances are already numeric vectors, so the usage of clustering is straightforward. In our example, we can consider triples for the performance, availability and cost values of architecture configurations. In the AC space, however, identifying adequate variables for the instance vector can be challenging, and several representation options might exist (e.g., complex features, or graph embeddings) [4]. In our example, let us assume that the configurations are represented by a triple $\langle device_1, device_2, device_3 \rangle$, which indicates the number of services allocated to the three devices. An additional aspect in the AC space is that the architectures are generally connected, depending on the transformations (or design decisions) made for transitioning from a given architecture to its neighbors. This interconnected structure of the data can influence the clustering process results.

When running the clustering process, there are a variety of technical parameters to be determined, such as: the specific clustering algorithm, the number of clusters (or a metric for choosing an appropriate number), and the distance function to compare instances. In this example, for each space we chose the number of clusters that minimized the *silhouette score*, and relied on the Euclidean distance to assess similarity among instances, according to standard ML guidelines for clustering [15].

Figs. 3a and 3b show the results of clustering the AC and QA spaces, respectively. Although these charts provide an intuition to the architect about groups of instances, partially contributing to Q1, Q2 and Q3; the charts have information overloading issues, and the architectural characteristics of each group are not easy to see. Furthermore, the links between the seven groups of configurations (Fig. 3b) and the five quality-attribute groups (Fig. 3a) are unclear. Thus, many architectural questions remain unanswered.

4 TAILORING CLUSTERING TO DESIGN

In this section, we showcase how the results of a pure clustering approach can be progressively simplified and enriched to convey architectural information about design decisions and quality-attribute tradeoffs (step ⑤ in Fig. 1). To this end, we introduce techniques for reducing the complexity of the spaces for our model problem.

4.1 Value discretization

To improve the architect's understanding of the instances in either the QA space or the AC space, we apply a *discretization* procedure that partitions the range of values for each quality attribute or architecture variable into an ordinal (or Likert-like) scale. For instance, for cost, we use a 5-point scale $\langle \text{very-cheap}, \text{cheap}, \text{average}, \text{expensive}, \text{very-expensive} \rangle$ which converts the numeric values for the attribute into categorical ones, as depicted in Fig. 4. For any instance in the QA space, we can then assign a label as a concatenation of the categorical values for the three quality attributes, which the architect can interpret as a quality-attribute tradeoff. An example of

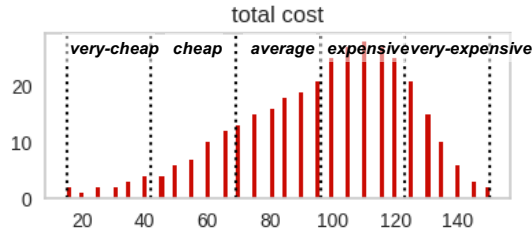


Figure 4: Example of discretization of values into categories.

such a label can be *very-slow/very-cheap/unreliable* for performance, cost and reliability, respectively. Similarly, for the possible configurations in our client-server style, we can apply a discretization of the architecture variables, which generates categorical values with respect to the number of services allocated to the devices. For example, we use the scale *<no-services, few-services, some-services, many-services>*, and then obtain labels such as *no-services/many-services/few-services* for the three available devices, respectively.

The main goal of the discretization is to reduce the combinations of values to a small set of labels. For the *QA* space, a 5-point scale results in 125 potential tradeoff categories; while for *AC* space, the 4-point scale results in 64 categories of architecture configurations. These represent information reductions of 37% and 19%, respectively. Alternative discretizations, such as quartiles, are possible.

4.2 Prototype selection and distances

Once clusters are identified and labels are assigned to their instances, we can simplify a space by selecting an instance that best reflects the cluster characteristics. In particular, we select a prototypical instance that is closest to the mean of the instances belonging to a cluster (also known as the *medoid* of the cluster). Furthermore, we use the label of the medoid as the label to characterize the whole cluster. Figs. 5b and 5a show how the spaces of Figs. 3a and 3b look like after condensing them by keeping only the cluster prototypes. These charts help architects to reason about *Q1* and *Q3*.

For the spatial layout of the figures, we rely on a *multi-dimensional scaling* (MDS) technique [5], which creates a 2D projection of the space that shows the dissimilarities among the prototypes by trying to preserve the distances between them (in the original space). In the *QA* space, the number of feasible tradeoffs has been now reduced to five categories (out of 125). In this way, the architect can clearly see that the most common tradeoff (0) is *very-fast/expensive/highly-reliable*, and other related tradeoffs are also well-represented in the space. Note also that tradeoff 3 is shown far part from the other tradeoffs, as its label *very-slow/very-cheap/unreliable* is quite different from the others. Other visualizations, such as parallel plots or radar charts, can also help to expose the feasible tradeoffs.

As a complement, Fig. 6 depicts the structural characteristics of the configurations for the five prototypes, highlighting the differences in the assignment of the services to devices with different hardware, and contributing to *Q2*. For instance, a visual inspection of the architecture of prototype 3 (at the top-right corner) corroborates a preference for cost over performance and reliability.

The notion of distance (or separation) among the prototypes should be chosen in such a way that it conveys an architectural meaning. Since the *QA* space often involves a handful of objectives, the Euclidean distance between the (numeric) quality-attribute values is a standard choice, and it was used in our example. For the

AC space, we interpret the distance between two configurations in terms of their delta of changes [2]. For simplicity, we considered the Euclidean distance between the architecture variables of the configurations; however, more-complex proposals can be explored. For instance, the architectural distance between a pair of configurations can be expressed in terms of the sequences of transformations (or decisions) applied on the initial architecture in order to reach those configurations, using a variant of the *hamming* distance [7].

Analogously to the condensed *QA* space, we can compute the prototypes for the clusters in the *AC* space, as shown in Fig. 5a. In this case, the feasible groups of configurations are simplified to 7 clusters (out of 64). An MDS projection has been applied to visually arrange the prototypes. Here, the architect can see that the configurations are evenly distributed across the groups, with some exceptions for clusters 5 and 6 being less represented than the rest. Note that, unlike Fig. 3a, the condensed clustering of architecture configurations preserves the edges from the original graph, which can help architects to identify the changes required to move from one configuration to another. Although this condensed view is useful for understanding the alternatives in *AC* space, if the architect would like to reason about the connections of these prototypes to the quality-attribute tradeoffs, that need is not supported.

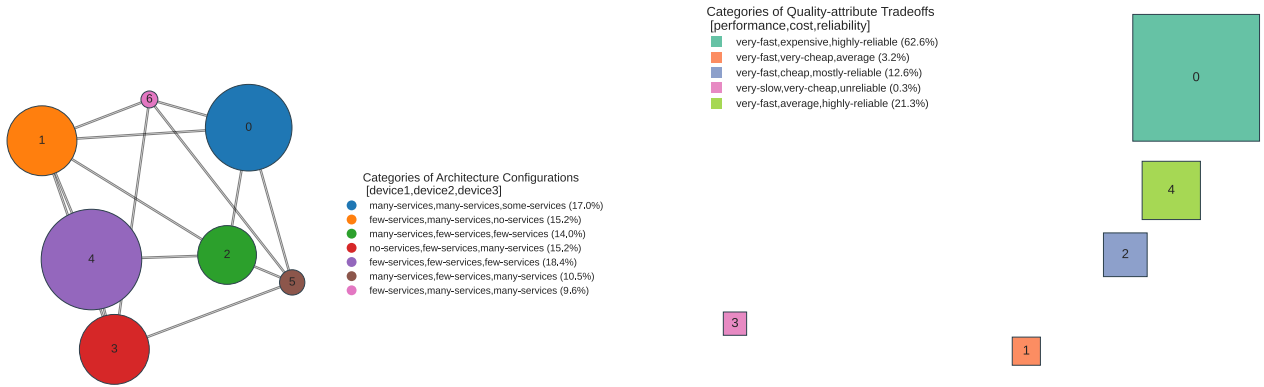
4.3 Correlation between the two spaces

The relations between the prototypes (and their underlying clusters) of the two spaces is seldom a one-to-one mapping. A group of architecture configurations will likely have heterogeneous quality-attribute characteristics, which means that subsets of instances can map to different quality-attribute tradeoffs. For instance, configurations with many services allocated to both the high-end and low-end devices but with few (or no) services allocated to the medium device, can have slight variations in their cost tradeoffs. To account for this situation, we need to check the mappings and eventually split the initial configuration clusters to ensure that the groups become homogeneous (with respect to their tradeoffs). When doing so, prototypes for the new groups need to be selected again. Fig. 7 shows the resulting clusters after performing the splitting process. The original clustering, based solely on the architecture configurations, is depicted by the shaded areas around subsets of graph nodes. Note that the *AC* space now has 21 prototypes. This number is larger than those for the condensed spaces, but is still small part (6%) of the initial design space generated by the tool.

This chart provides a more detailed view of the (whole) space, and helps to address *Q2*, *Q4*, *A5* and *Q5*. The new prototypes attempt to be representative but also diverse with respect to both architectural structure and quality-attribute tradeoffs. As in previous graphs, the edges denote paths to transition from one (type of) configuration to another, but we also expose how the tradeoffs might change due to those transitions. As mentioned in section 4.2, the distance between prototypes indicates their proximity in terms of architectural changes. The architect can further inspect the architecture behind any prototype, as in Fig. 6.

5 RELATED WORK

Several tools for automated architecture optimization that generate a set of alternatives have been proposed [1, 2]. These tools work



(a) Condensed view of the space of architecture configurations. Only cluster prototypes are shown. Node sizes are proportional to the number of cluster instances. Edges indicate architecture transformations to move among clusters.

(b) Condensed view of the quality-attribute space. Only cluster prototypes are shown. Square sizes are proportional to the number of cluster instances.

Figure 5: Improved clustering of the two spaces (support for Q1 and Q3).

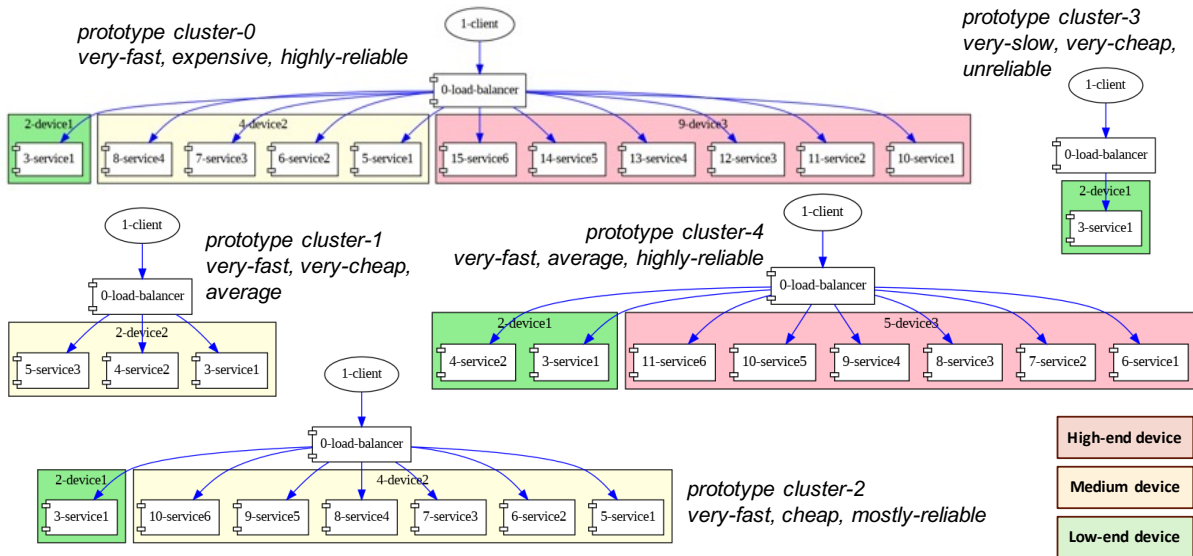


Figure 6: Architecture configurations of the 5 prototypes for the (condensed) quality-attribute space (support for Q2).

mostly as black boxes, and their internal search space is not comprehensible by humans. Recent approaches, like *SQuAT-Viz* [8] and *Voyager* [11], have investigated visualization techniques for helping architects to understand tradeoffs, and have also evaluated their usability. Among other techniques, *SQuAT-Viz* [8] uses radar charts and scatter plots for the QA space, showing all possible combinations of tradeoffs. *Voyager* [11], in turn, combines tradeoff analysis along with architectural structure visualizations, aiming to connect these two spaces, which is a concern shared by our framework. However, it does not consider space reduction issues.

Other authors have attempted to explain tradeoff spaces using dimensionality reduction and clustering techniques. Camara et al. [4] propose PCA (Principal Component Analysis) loading plots to relate quality-attribute and architectural variables. In the planning domain, Wohlrab et al. [14] complement the previous PCA plots with clustering and decision trees. The usage of 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 PCA plots, which often implies some information loss when going to a 2D representation.

The *GATSE* tool [12] 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 QA space to narrow down or deepen the search for alternatives.

Kinneer and Herzig [9] 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 is proposed to group the architectures

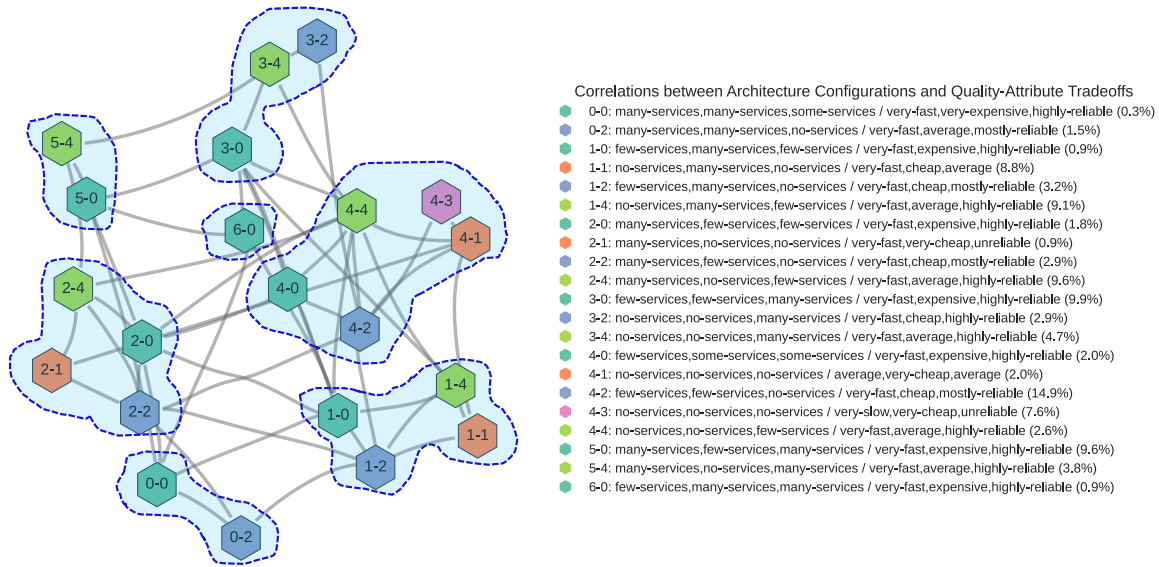


Figure 7: Combining prototypes for the architecture configurations and quality-attribute spaces (support for Q2, Q4, Q5 and Q6).

and select a representative instance from each group. The clustering is tied to the notion of architecture distance, like in our framework.

Question (or need)	Basic clustering	Condensed (tailored) clustering			Inspection of configuration(s)	Charts (Figures)	
		Discretization	Prototype selection	Distance & layout			Combination of the 2 spaces
Q1	⊕	✓	✓	✓	n/a	n/a	4a, 5b
Q2	⊕	✓	✓	n/a	✓	✓	4a, 6, 7
Q3	⊕	⊕	✓	✓	n/a	✓	4a, 5a
Q4	×	⊕	✓	⊕	✓	⊕	7
Q5	×	⊕	✓	✓	⊕	⊕	7
Q6	×	⊕	✓	✓	⊕	⊕	7

✓ Good support ⊕ Partial support × No support

Figure 8: Summary of contributions of the different techniques and charts to the driving architectural questions.

6 CONCLUSIONS AND PERSPECTIVE

In this work, we proposed a framework for making automated design exploration more effective from the architect’s point of view. In particular, we argue that the process should be driven by architectural questions. We exercised this framework using clustering and other related techniques¹. Fig. 8 gives a summary of how those techniques supported our initial set of questions. Naturally, including more questions might require additional techniques. For instance, the architect could ask about the key design decisions behind particular architecture configurations. This type of question focuses on the edges of the graph (AC space) rather than on the nodes. Furthermore, we foresee that questions that require combined information from the AC and QA spaces can be challenging. Making progress on the questions and supporting architect’s interactions will serve us to improve the workflow for the proposed framework (Fig. 1).

As future work, we plan to enhance our repertoire of questions and techniques, providing a better integration among them, for instance, by means of storytelling mechanisms. User studies will be also needed as important instruments to validate our tooling effort. Additionally, we will investigate the development of a design assistant for the framework using generative AI.

¹ Colab notebook with the techniques and visualizations: <https://shorturl.at/fnrw6>.

REFERENCES

- ALETI, A., BUHNOVA, B., GRUNSKÉ, L., KOZIOLEK, A., AND MEEDENIYA, I. Software architecture optimization methods: A systematic literature review. *IEEE Trans. on Soft. Eng.* 39, 5 (2013), 658–683.
- ARCELLI, D., CORTELESSA, V., D’EMIDIO, M., AND DI POMPEO, D. Easier: An evolutionary approach for multi-objective software architecture refactoring. In *2018 IEEE Int. Conf. on Software Architecture (ICSA)* (2018), pp. 105–115.
- BASS, L., CLEMENTS, P., AND KAZMAN, R. *Software Architecture in Practice*. SEI series in software engineering. Addison-Wesley, 2003.
- CÁMARA, J., SILVA, M., GARLAN, D., AND SCHMERL, B. Explaining architectural design tradeoff spaces: A machine learning approach. In *Software Architecture: 15th European Conf., ECSA 2021, Sweden, Proceedings* (Berlin, Heidelberg, 2021), Springer-Verlag, p. 49–65.
- COX, M. A. A., AND COX, T. F. *Multidimensional Scaling*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 315–347.
- CÁMARA, J., WOHLRAB, R., GARLAN, D., AND SCHMERL, B. Focusing on what matters: Explaining quality tradeoffs in software-intensive systems via dimensionality reduction. *IEEE Software* (2023), 1–10.
- DIAZ-PACE, J. A., WOHLRAB, R., AND GARLAN, D. Supporting the exploration of quality attribute tradeoffs in large design spaces. In *Software Architecture* (Cham, 2023), B. Tekinerdogan, C. Trubiani, C. Tibermacine, P. Scandurra, and C. E. Cuesta, Eds., Springer Nature Switzerland, pp. 3–19.
- FRANK, S., AND VAN HOORN, A. Squat-vis: Visualization and interaction in software architecture optimization. In *Software Architecture - 14th European Conf., ECSA 2020 Tracks and Workshops, Proc.* (2020), vol. 1269, Springer, pp. 107–119.
- KINNEER, C., AND 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* (New York, NY, USA, 2018), MODELS ’18, ACM, p. 392–402.
- KWIATKOWSKA, M., NORMAN, G., AND PARKER, D. PRISM 4.0: Verification of probabilistic real-time systems. In *Proc. 23rd Int. Conf. on Computer Aided Verification (CAV’11)* (2011), vol. 6806 of LNCS, Springer, pp. 585–591.
- MASHINCHI, J., AND CÁMARA, J. Voyager: Software architecture trade-off explorer. In *Software Architecture - 14th European Conf., ECSA 2020 Tracks and Workshops, Proceedings* (2020), vol. 1269, Springer, pp. 55–67.
- PROCTER, S., AND WRAGE, L. Guided architecture trade space exploration: Fusing model based engineering and design by shopping. In *2019 ACM/IEEE 22nd Int. Conf. on Model Driven Eng., Languages and Systems (MODELS)* (2019), pp. 117–127.
- QUESADA, A. R., ROMERO, J. R., AND VENTURA, S. Interactive multi-objective evolutionary optimization of software architectures. *Inf. Sci.* 463-464 (2018).
- WOHLRAB, R., CÁMARA, J., GARLAN, D., AND SCHMERL, B. Explaining quality attribute tradeoffs in automated planning for self-adaptive systems. *Journal of Systems and Software* 198 (2023), 111538.
- XU, D., AND TIAN, Y. A comprehensive survey of clustering algorithms. *Annals of Data Science* 2 (2015), 165 – 193.