# Focusing on *What Matters*: Explaining Quality Tradeoffs in Software-Intensive Systems via Dimensionality Reduction

**Javier Cámara**
ITIS Software — Universidad de Málaga, Spain

**Rebekka Wohlrab**
Chalmers — Univesity of Gothenburg, Sweden

**David Garlan**
Carnegie Mellon University, USA

**Bradley Schmerl**
Carnegie Mellon University, USA

***Abstract*—Building and operating software-intensive systems often involves exploring decision spaces made up of large numbers of variables and complex relations among them. Understanding such spaces is often overwhelming to human decision makers, who have limited capacity to digest large amounts of information, making it difficult to distinguish the forest through the trees. In this article, we report on our experience in which we used dimensionality reduction techniques to enable decision makers in different domains (software architecture, smart manufacturing, automated planning for service robots) to focus on the elements of the decision space that explain most of the quality variation, filtering out noise, and thus reducing cognitive complexity.**

**DEVELOPING** and operating complex software-intensive systems involves making a large number of decisions. The most important ones are often focused on quality attributes such as performance, security, energy efficiency, or reliability, and typically require answering questions such as: What response time is acceptable? What level of security is needed? How reliable does the system need to be? How much may the computing resources cost?

Many decisions involve tradeoffs among quality attributes because it is often not possible to satisfy conflicting qualities, e.g., to have a high-performing system with high security at low cost. In practice, many of those tradeoffs are often poorly understood by stakeholders (e.g., decision makers such as software architects) because decision spaces involve a large number of variables, and information about their relations and overall influence on qualities (i.e., to what extent a given variable or choice can influence the different quality attributes) is difficult to tease out [6], [7].

Decision makers need tools and techniques to help them understand the tradeoffs of complex decision spaces and guide them to good choices, enabling them to answer questions such as: Why are these tradeoffs being made, and not

Published by the IEEE Computer Society

others? What are the most important variables and qualities that are driving the key decisions? How sensitive is the satisfaction of constraints or the achievement of optimality to a particular set of decisions? Which choices are correlated with others, either positively or negatively?

To help decision makers understand quality tradeoffs and make better informed decisions, we built approaches to explaining decision spaces from the perspective of quality tradeoffs in different kinds of software-intensive systems. We relied on standard techniques that involve dimensionality reduction, such as Principal Component Analysis (PCA) [8] and Multiple Correspondence Analysis (MCA) [13], [10], which can identify the variables that are the chief contributors to variability along quality dimensions, and tease out their relations (e.g., cost is negatively correlated with the selection of a given software component, while performance is positively correlated with it). Using these techniques enables decision makers to focus on the relevant pieces of information, thus reducing cognitive complexity [12]. Dimensionality reduction techniques on their own, however, are not enough to understand the impact of specific decisions on qualities. Hence, we also complemented our approaches with techniques such as Decision Tree Learning (DTL) which enables identifying the impact of key decisions on quality attributes (e.g., those described in standards such as [ISO 25010]).

In this article, we describe our experience applying the tradeoff-focused explainability techniques that we developed to three areas within development and operation of software-intensive systems, namely design of component-and-connector software architectures [6], [7], automated planning for autonomous service robots [14], and smart manufacturing. Drawing upon that experience, we distill a set of lessons learned, propose some general guidelines for researchers and practitioners to apply analogous techniques in the context of their work, and discuss use cases for this class of approach that remain to be explored.

## Decision Spaces in Software-Intensive Systems

Complex decision spaces can emerge across many aspects of the development and operation of software-intensive systems. Here, we explore three domains.

### Software Architecture

During development, architects have to face design decisions that often involve selecting and composing loosely coupled, pre-existing components or services that have different attributes (e.g., performance, reliability, cost).

There are multiple approaches that help to automate the search for good architecture designs and rely on a variety of techniques such as stochastic search, Pareto analysis, and quantitative verification [1]. Despite being informative, these approaches do not always clarify why and how architectural configurations were selected because they do not explicitly link design decisions to the satisfaction of requirements and they yield vast amounts of data that are not easy to interpret.

Consider for instance Figure 1a, which shows the analysis results of a service-based system obtained by applying one such technique using our tools as described in [4], [5]. In this system, design decisions involve selecting among multiple functionally equivalent service implementations with different qualities (cost, reliability, and performance), setting thresholds for parameters (timeouts, service invocation retries), and determining the topological arrangement of the service composition (e.g., should multiple services of the same type be invoked, or just one? if the former, should they be invoked sequentially, or in parallel?). The plot shows the minimized cost of configurations for different levels of constraints on response time and reliability.

This plot conveys the intuition that higher response times and lower reliability correspond to lower costs, whereas peaks in cost are reached with the lowest failure rates and response times.

Although these approaches are informative and can help architects to understand what specific configurations might work well in a given situation, they do not facilitate understanding what design decisions influenced these tradeoffs. Determining to what extent improvements on qualities are a function of the choice of a specific service implementation, the topological arrangement of the composition, or the value of configuration parameters is something that cannot be addressed with such existing approaches.

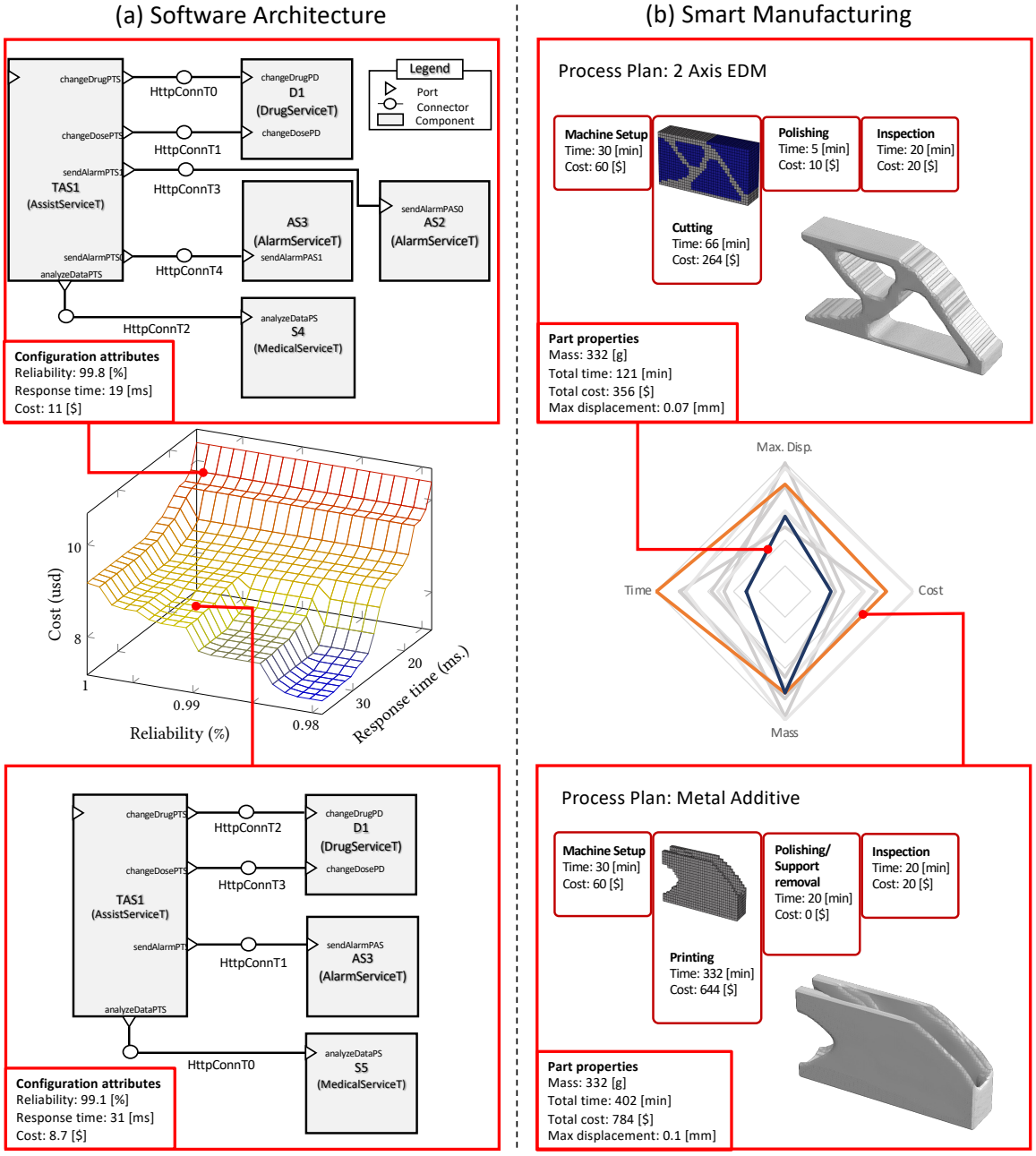


**Figure 1.** Quality tradeoff spaces: (a) software architecture, (b) smart manufacturing.

### Robot Mission Planning

Robotic systems often rely on mission planning components that plan an optimal high-level mission such as the navigation route for a mobile robot. Different plans may result in different tradeoffs between properties such as travel time, safety, and privacy.

There exists a variety of mission planning algorithms and solutions. A problem for human end users is that for realistic maps, it is not always trivial to see why a certain plan was chosen and is optimal with respect to a given set of quality attributes. Human end users might be asked to indicate the priorities of quality attributes, but cannot always see how the prioritization affects the generated plans.

Although some existing approaches for mission planning support an investigation of the

tradeoffs among quality attributes (e.g.,[11]), they do not streamline the process to let the decision maker focus on key decisions and systematically investigate how quality attributes impact each other in detail.

**Company and Open Source Contexts**

We collected data from several open source and industrial projects. For the software architecture example, we used the open-source Tele Assistance System (TAS)[a]. The open-source system is based on a real-world service-based system. It has been developed by members of the research community on self-adaptive systems and used for various example purposes.

For robot mission planning, we have used examples in which robotic systems optimize their travel plans for multiple quality attributes such as a robot in a building on campus. We also draw on our experience in projects such as the DARPA Building Resource Adaptive Software System's (BRASS) program.[b]

For the example of smart manufacturing, we have been collaborating with the company Lockheed Martin[c] over the course of several years through the Manufacturing Futures initiative at Carnegie Mellon University.

[a] https://www.hpi.uni-potsdam.de/giese/public/selfadapt/exemplars/tas/

[b] http://www.cs.cmu.edu/~brassmars/

[c] https://www.lockheedmartin.com/en-us/capabilities/advanced-manufacturing.html

### Smart Manufacturing

When manufacturing three-dimensional objects (e.g., a bracket for a bicycle), it is common that multiple manufacturing techniques can be used to produce parts that satisfy a set of physical requirements. For example, it is possible to use *additive manufacturing* (a technique similar to a 3D printer), and *subtractive manufacturing* (in which parts are removed from a block of material) to arrive at the final product. The two techniques require different materials in powder form or in a block, and they come with different tradeoffs. In Figure 1b, the analysis of process plans generated by a CAD prototype tool for smart manufacturing is shown. The one at the top is a 2-axis machining process plan that relies on subtractive manufacturing, whereas the process plan at the bottom is a process plan for metal additive manufacturing. These two plans are two examples in the manufacturing design space that is shown at the center. The process plan at the top comes with a lower cost, lead time, and maximum displacement than the additive process plan at the bottom.

Also, in practice, large production companies often collaborate with multiple suppliers who can manufacture customized parts. For example, some techniques lead to a more robust product than others. Different suppliers can take more or less time, have different cost models, and different sets of machines. Manufacturers now struggle with understanding how different production options affect tradeoffs among qualities such robustness, time and cost.

## Our Approach: Quality Tradeoff Explanation via Dimensionality Reduction

When making decisions about which architecture or plan to choose that trade-off different qualities in a given domain, a decision-maker must understand the consequences of those decisions, but they must focus on the qualities that are most important. To do this, we propose quality tradeoff explanation via dimensionality reduction (Figure 2), an approach in which problem descriptions jointly specified by domain experts and decision makers are employed to produce explanations of quality tradeoffs (solution classes, decision and quality variable relations and influence on solution variability, identification and impact of key decisions on quality attributes) that contribute to a better understanding of the decision space.

Domain experts (persons who possess knowledge in a particular area not necessarily available to decision makers) produce problem specifications that capture the main concepts of the domain and their relations (input I1) such as software components and connectors in a software architecture, or robot/world states and actions in mission planning. Problem specifications are complemented by a set of quantifiable quality attributes (I2) that capture different dimensions

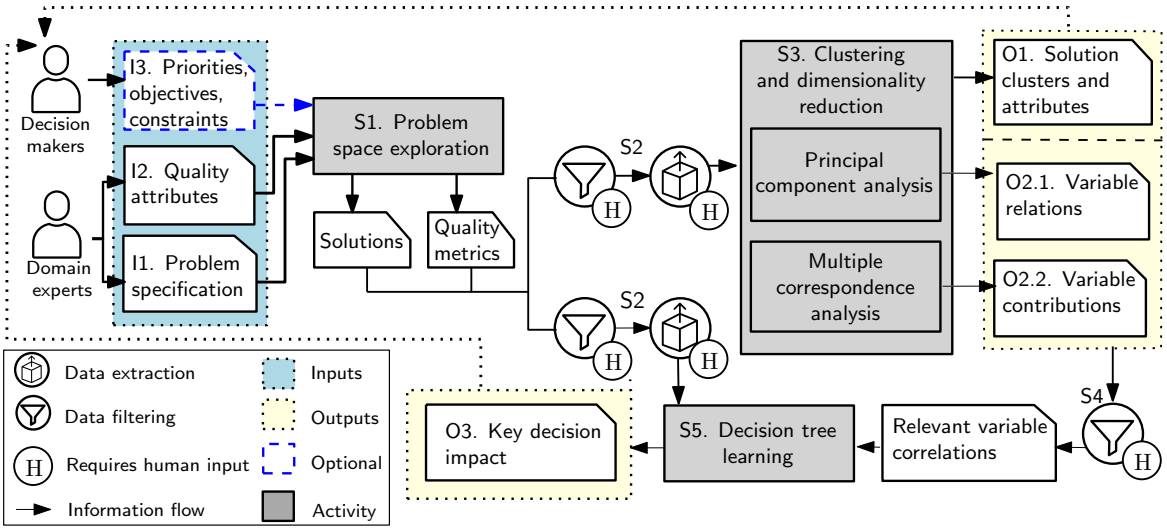


**Figure 2.** A diagram illustrating the quality tradeoff explanation approach.

of concern in the quality space, such as cost, performance, and safety. An additional, optional input to the process (I3) is provided by decision makers and consists of a set of quality objectives and constraints that result from the formalization of quantitative requirements on quality attributes (e.g., "*maximum response time must be under 1000ms*", or "*the energy consumed during the robot's mission should be minimized*"). In some cases, e.g., when objectives are defined in terms of utility maximization, these can be accompanied by a specification of the priorities of decision makers, which capture the relative importance of the satisfaction of constraints and objectives. Our approach, however, is agnostic to the decision-making strategy, which can make use of alternatives such as utility theory (applied in our work on software architecture and automated planning), and *satisficing*, i.e., choosing solutions that achieve at least a minimum level of acceptability (applied in our work on smart manufacturing).

### Problem Space Exploration

All the inputs (I1-I3) are employed by problem space exploration (S1), the first stage of our approach in which a set of solutions is automatically generated from problem specifications. The strategies and techniques employed to explore the problem space can vary depending on the situation, but two key requirements for them are that they: (i) must be able to produce a significant number of solutions to enable building a large enough dataset (e.g., to comply with the *ten times rule* [2]), and (ii) must be able to quantify the quality of solutions along the different dimensions of concern. Although multiple techniques can satisfy these requirements, automated approaches to solution generation based on quantitative formal verification have worked well when applying our approach to several areas. In our work on architectural space exploration [7], we employed techniques that combine structural synthesis and quantitative verification to analyze quantitative formal guarantees across the architectural design space [4], [5], whereas in our work on automated planning [14] we employed the probabilistic model checker PRISM [9] to synthesize plans for service robot missions as policies for Markov decision processes. For smart manufacturing we use automated computer-aided design (CAD) and computer-aided manufacturing (CAM) tools, coupled with automated schedulers.

### Data Extraction and Filtering

Our approach requires data filtering and extraction at different stages of the process (S2, S4). In S2, filtering is required to select a relevant subset of variables and leave out characteristics of solutions and quality metrics that are not of concern to the problem at hand, whereas data extraction is related to how information from solutions and their qualities can be repre-

sented as data points in *data frames* suitable for use with clustering and dimensionality reduction techniques (S3), as well as with DTL (S5). In architecture design, for instance, solutions (which correspond to architectural configurations) have to be encoded in dataframe rows as a set of variables that capture, e.g., the presence of specific components in the configuration and their properties, the existence of attachments between specific component ports, or the value of configuration parameters. In contrast, in automated planning, plans for service robot missions have to be encoded e.g., in terms of which actions (or sequences thereof) are employed by plans, reachable world states, and so forth. Filtering and extraction require awareness about the semantics of the data being manipulated, and hence require input both from domain experts and decision makers. Filtering is also used later in the process (S4) to identify relevant variable correlations for DTL, as discussed below.

### Clustering and Dimensionality Reduction

In high-dimensional spaces, dimensionality reduction techniques like PCA [8] can be leveraged to identify correlations between variables in large datasets and indicate how variables contribute to the differences in solutions. The technique involves computing so-called *principal components* (PC), which are linear combinations of the original variables that explain the variance in the data in decreasing order (i.e., the first PC or PC1 carries the most information regarding differences between samples, PC2 explains the most variance not covered by PC1, and so on). The output of PCA includes: (i) the percentage of total *variance* of the dataset explained by each PC, (ii) correlation *loadings*, which describe how much variables contribute to explained variance, as well as their correlations, and (iii) *scores*, describing properties of the samples. In our approach, we are mainly concerned with variances and loadings, focusing on how much information is explained by the principal components (O2.2) and how variables are related to each other (O2.1). In some cases, Multiple Correspondence Analysis (MCA) can complement or be used as an alternative to PCA when most of the variables are of a categorical nature, although numerical variables are also supported. Figure 3a shows a PCA correlation loadings plot for the system in Figure 1a, where points in the same quadrant represent positively correlated variables (e.g., cost, reliability, and the selection of service implementation AS3), whereas points that form an angle greater than 90 degrees are negatively correlated (e.g., selecting AS3 with response time).

Clustering is concerned with making sense of data by categorizing data points (i.e., solutions) into collections that share similarities (O1). Our approach employs a widely used clustering technique (*k-means clustering*) that involves discerning $k$ clusters, and in which each data point is allocated to the cluster with the nearest mean. We apply k-means clustering in combination with PCA and MCA [13], [10] for retaining as much variance as possible in as few dimensions as possible, while calculating and describing clusters of solutions in the data. Reasoning about solutions in terms of clusters is helpful for humans to understand patterns and characteristics of different categories of solution. Figure 3b shows an example plot depicting different clusters of robot mission plans [14], where C1 prioritizes privacy, C2 prioritizes safety and collision avoidance, and C3 prioritizes travel time at the expense of other quality attributes.

### Decision Tree Learning

Decision tree learning [3] (S5) is a supervised learning technique used in statistics, data mining, and machine learning that allows predicting the value of a target variable selected by the user, based on other variables' values. DTL can be used to grow both classification trees (to predict categorical values) and regression trees (to predict numerical values), and is particularly useful in contexts that involve complex datasets with high dimensionality and heterogeneous data types. In our approach, we leverage the insights provided by PCA/MCA to inform the target variable selection (S4) because variables that are of relevance to explain the variance in the data are good candidates as target variables to explain the impact on qualities of choices associated with key decisions. DTL receives as input both the data frame produced in S2 and the set of relevant variables identified in S4, and produces a set of decision trees that link concrete threshold values of design variables with impacts on qualities (O3)

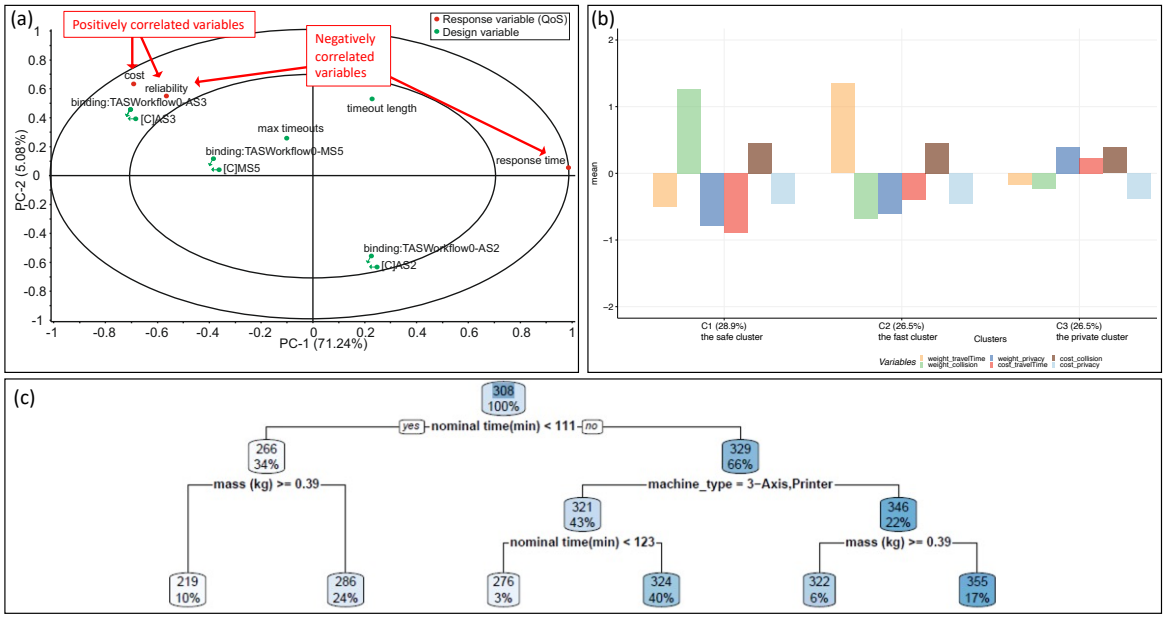


**Figure 3.** Example plots showing: (a) a PCA correlation loadings plot from the service-based system shown in Figure 1, where the most relevant variables are shown in the outer ellipse (adapted from [7]); (b) clusters of robot mission plans (adapted from [14]); and (c) a decision tree predicting cost for smart manufacturing.

that decision makers can use to inform their decisions.

Figure 3c shows a decision tree for the smart manufacturing example that predicts the cost of the manufactured part. It shows that if the nominal time is high and the used machine is not a 3-axis or printer, the cost is at its highest level. On the contrary, cost is at its lowest if the manufacturing process is fast and the generated mass is high. These insights indicate that the most important variable for cost is the time that the production process takes: the faster the process, the cheaper the part.

### Closing the Loop: Iterative Use of the Approach

The explanation artifacts produced as outputs of the approach are to be consumed by decision makers, who can then employ newly acquired insights about quality tradeoffs to refine their inputs (i.e., objectives, constraints, priorities) in subsequent iterations of the process. For example, if we look at Figure 3c, we can observe that cost is lowest if the manufacturing process is fast and the generated mass is high. If the decision maker wants to minimize cost, they can look at these plots and change the input parameters to focus on low-cost solutions in further iterations.

## Lessons Learned when Applying Explainability Techniques

Our lessons learned focus on (i) solution generation, (ii) information needs that can be met using our approach, and (iii) concerns when applying the approach in a real-world setting with changing needs.

### Solution Generation

The use of our explainability approach is largely facilitated by the improving mechanisms for automatic space exploration (step S1 in Figure 2) such as automated planners and design generators. We observed that the more complete the space exploration is, the better the results are. Consequently the chief performance cost is in the generation of points in the design space, and not running the explainability techniques.

When generating solutions for different applications, we might have different variables and datasets depending on the aspects we want to investigate. They could be connected to priorities of different quality attributes or to the general properties of the solution domain. In our examples, we focused on quantifiable quality attributes, which implied that PCA was an adequate technique. When dealing with mostly qualitative

quality attributes, such as usability, it might be necessary to rethink how data can be represented with categorical variables and focus on MCA.

Depending on the decision makers' interests and design processes, the approach can be used to start with a narrow space that can be expanded if needed or on a large space that can be iteratively restricted. Those two approaches require different methodologies. For example, when working in the domain of manufacturing, we found that requirements may initially be unclear and stakeholders need to hypothesize what are important variables. In this case our approach becomes a useful tool to explore and refine requirements.

### Information Needs

Information needs at different points in time might change. In initial phases, decision makers can use our approach to understand what properties of solutions make a difference to the outcomes. Decision makers might wonder: What decisions do (not) matter? How does a variable impact others?

In later phases, as designers explore a design space, they might have thresholds or constraints in mind. Decision tree learning is particularly useful for exploring those thresholds and analyzing the feasibility of constraints. For practitioners, it can be difficult to tell which constraints are crucial and which ones can be relaxed. Depending on what properties of the problem space a stakeholder cares about, their solution space might be different. It is often desirable to explore the design space and problem space at the same time.

### Real-world Applications and Changing Needs

In most real-world settings there is a cost of change. Our approach accounts for change by supporting the interactive exploration of solution spaces. When selecting solutions, it can be useful to keep changing needs in mind and explore alternatives to a given solution. For example, if we consider a given manufacturing process plan, it can be desirable for operators not to change the machine settings too much, but operate similar machine types and processes. This concern should be taken into account when exploring solution spaces and thresholds using decision tree learning. Choosing a solution that is close to a threshold may imply a radical change to the design will be needed if requirements or information (e.g., the cost of materials for manufacturing) changes.

Changing constraints or requirements comes at a cost. To generate solutions, we ran algorithms that employ, among other things, the priorities of quality attributes. Depending on what quality attributes are considered important, different solutions are obtained. It would be necessary to rerun the space exploration if a constraint or quality attribute were removed. As an alternative strategy, one could sample different combinations of parameters and not encode the priorities of quality attributes. That strategy would lead to potentially suboptimal solutions but a broader exploration of possible solutions. The selection of how broadly one wants to sample depends on the questions that the stakeholders want to get answers to. If we sample only optimal plans, we can understand how the priorities of quality attributes impact those plans. If we sample also suboptimal plans, we can understand the general tradeoffs within the solution space.

## Focusing on *What Matters*

Decision making during the construction and operation of software-intensive systems is a complex task that involves exploring large quality tradeoff spaces made up of large numbers of variables and complex relations among them. Understanding such decision spaces is often overwhelming to human decision makers, who have limited capacity to digest large amounts of information, making it difficult to distinguish the forest through the trees. In this article, we have reported on our experience in which we used dimensionality reduction techniques to enable decision makers to focus on *what matters*, i.e., on the elements of the decision space that explain most of the variation, filtering out noise, and thus reducing cognitive complexity. This represents a meaningful step forward because existing approaches to analyze quality tradeoffs in decision spaces (e.g., in software architecture) were able to yield large sets of (potentially optimal) point solutions, but they could not provide the kind of insights that decision makers can obtain when the main variables, relations, classes of solution, and impact of decisions on qualities are clarified and dissected for them. There is still a long road ahead in which, beyond further evaluation

to validate the performance and user experience of our proposal, we envisage the extension of this class of approach to include more intuitive domain-specific visualizations, to be able to study evolving phenomena (with application to e.g., autonomous systems), and to deal with highly non-linear systems. We believe that pursuing approaches that share the same philosophy can provide a significant contribution to many professionals in future software development endeavors.

## ACKNOWLEDGMENTS

This work was partially funded by the Spanish Government (FEDER/Ministerio de Ciencia e Innovación–Agencia Estatal de Investigación) under projects TED2021-130523B-I00 and PID2021-125527NB-I00, supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation, N00014172899 from the Office of Naval Research, the NSA under Award No. H9823018D0008, and Lockheed-Martin.

In particular we would like to thank Levent Burak Kara, Greg Federer, Steve Smith, Zack Rubenstein, Hongrui Chen, Aditya Joglekar, and Ani Kimmel of Carnegie Mellon as colleagues who worked with us on the smart manufacturing project, and whose work is depicted in Figure 1(b).

## REFERENCES

1. Aldeida Aleti, Barbora Buhnova, Lars Grunske, Anne Koziolek, and Indika Meedeniya. Software architecture optimization methods: A systematic literature review. *IEEE Transactions on Software Engineering*, 39(5):658–683, 2012.
2. Ahmad Alwosheel, Sander van Cranenburgh, and Caspar G Chorus. Is your dataset big enough? sample size requirements when using artificial neural networks for discrete choice analysis. *Journal of choice modelling*, 28:167–182, 2018.
3. Leo Breiman, Jerome H Friedman, Richard A Olshen, and Charles J Stone. *Classification and regression trees*. Routledge, 2017.
4. Javier Cámara. HaiQ: Synthesis of software design spaces with structural and probabilistic guarantees. In *FormaliSE@ICSE 2020: 8th International Conference on Formal Methods in Software Engineering*, pages 22–33. ACM, 2020.
5. Javier Cámara, David Garlan, and Bradley Schmerl. Synthesizing tradeoff spaces with quantitative guarantees for families of software systems. *J. Syst. Softw.*, 152:33–49, 2019.
6. Javier Cámara, Mariana Silva, David Garlan, and Bradley Schmerl. Explaining architectural design tradeoff spaces: A machine learning approach. In *Proc. of the 15th European Conference on Software Architecture (ECSA 2021)*, pages 49–65. Springer, 2021.
7. Javier Cámara, Rebekka Wohlrab, David Garlan, and Bradley Schmerl. Extra: Explaining architectural design tradeoff spaces via dimensionality reduction. *Journal of Systems and Software*, 198:111578, 2023.
8. Ian T Jolliffe. Principal components in regression analysis. In *Principal component analysis*, pages 129–155. Springer, 1986.
9. Marta Kwiatkowska, Gethin Norman, and David Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *Computer Aided Verification*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.
10. Angelos Markos, Alfonso Iodice D'Enza, and Michel van de Velden. Beyond tandem analysis: Joint dimension reduction and clustering in r. *Journal of Statistical Software*, 91(10), 2019.
11. Roykrong Sukkerd, Reid Simmons, and David Garlan. Tradeoff-focused contrastive explanation for mdp planning. In *2020 29th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*, pages 1041–1048. IEEE, 2020.
12. John C Thomas and John T Richards. Achieving psychological simplicity: Measures and methods to reduce cognitive complexity. In *Human-Computer Interaction*, pages 179–198. CRC Press, 2009.
13. Michel van de Velden, Alfonso Iodice D'Enza, and Angelos Markos. Distance-based clustering of mixed data. *Wiley Interdisciplinary Reviews: Computational Statistics*, 11(3):e1456, 2019.
14. Rebekka Wohlrab, Javier Cámara, David Garlan, and Bradley Schmerl. Explaining quality attribute tradeoffs in automated planning for self-adaptive systems. *Journal of Systems and Software*, 198:111538, 2023.

**Javier Cámara** is Associate Professor of Computer Science at the University of Málaga and Honorary Visiting Fellow at the Department of Computer Science, University of York. His research interests include self-adaptive and autonomous systems, software architecture, formal methods, as well as cyber-physical and AI systems. He received a European Ph.D. degree with honors from the University of

Málaga in 2009. Contact him at jcamara@uma.es.

**Rebekka Wohlrab** is an assistant professor at Chalmers University of Technology in Sweden and an adjunct faculty at Carnegie Mellon University. She received her Ph.D. from Chalmers University of Technology. Her research interests include requirements engineering and software architecture, especially in connection with human-on-the-loop autonomous systems. Contact her at wohlrab@chalmers.se.

**David Garlan** is Professor of Computer Science and Associate Dean in the School of Computer Science at Carnegie Mellon University. His research interests include software architecture, self-adaptive and autonomous systems, and formal methods. He has received a Stevens Award Citation for "fundamental contributions to the development and understanding of software architecture as a discipline in software engineering," an Outstanding Research award from ACM SIGSOFT for "significant and lasting software engineering research contributions through the development and promotion of software architecture," an Allen Newell Award for Research Excellence, an IEEE TCSE Distinguished Education Award, and a Nancy Mead Award for Excellence in Software Engineering Education. He is a Fellow of the IEEE and ACM. Contact him at garlan@cs.cmu.edu.

**Bradley Schmerl** is a principal systems scientist in the School of Computer Science at Carnegie Mellon University, Pittsburgh. His research interests include software architecture, self-adaptive systems, and software engineering tools. Schmerl received a Ph.D. in computer science from Flinders University, Adelaide, Australia. Contact him at schmerl@cs.cmu.edu.