GuideArch: Guiding the Exploration of Architectural Solution Space under Uncertainty

Naeem Esfahani
Department of Computer Science
George Mason University
Fairfax, Virginia, USA
nesfaha2@gmu.edu

Sam Malek
Department of Computer Science
George Mason University
Fairfax, Virginia, USA
smalek@gmu.edu

Kaveh Razavi
Department of Computer Science
George Mason University
Fairfax, Virginia, USA
srazavi2@gmu.edu

***Abstract*—A system's early architectural decisions impact its properties (e.g., scalability, dependability) as well as stakeholder concerns (e.g., cost, time to delivery). Choices made early on are both difficult and costly to change, and thus it is paramount that the engineer gets them *"right"*. This leads to a paradox, as in early design, the engineer is often forced to make these decisions under uncertainty, i.e., not knowing the precise impact of those decisions on the various concerns. How could the engineer make the *"right"* choices in such circumstances? This is precisely the question we have tackled in this paper. We present *GuideArch*, a framework aimed at quantitative exploration of the architectural solution space under uncertainty. It provides techniques founded on fuzzy math that help the engineer with making informed decisions.**

***Index Terms*—Software Architecture, Uncertainty, Decision Making**

I. Introduction

A software system's *early architecture* is the set of principal decisions made at the outset of a software engineering project. Early architecture encompasses choices at application, system, and hardware level that could have an impact on the software system's properties. A candidate architecture results from selecting a viable alternative for each and every decision. A common practice is to carefully assess the system's early architecture for its ability to satisfy functional and non-functional requirements, as well as other stakeholder concerns, such as cost and time to delivery.

Early architectural decisions are crucial, as they determine the scope of capabilities and options that can be exercised later in the system's life cycle. Given the crucial impact of early architectural decisions on the system's properties, changing them in subsequent phases of the engineering process are often both difficult and costly. At the same time, making such decisions is a complex task mired with lots of uncertainty. Getting them "wrong" poses a risk to any software engineering project.

One of the major thrusts of the software engineering research has been to transform the process of making such decisions from an art form exercised successfully by a select few to a repeatable process guided through scientific reasoning and formal analysis. A few notable examples include ATAM [5], CBAM [13], and ArchDesigner [1]. Such efforts have not aimed to replace the engineer's experience and knowledge, but to rather augment it through provisioning of appropriate methods and tools.

While great strides have been made, the existing approaches do not provide adequate support for dealing with uncertainty in early architecture [11]. In fact, there is no quantitative method of even comparing two architectures under uncertainty, let alone selecting the "right" architecture from the many possible candidates [1], [11].

In this paper, we describe GuideArch, a quantitative framework and accompanying tool aimed at guiding the exploration of architectural solution space under uncertainty. It allows the architect to make informed decisions using imperfect information. This alleviates the architect from manually sifting through an often large solution space, and instead allows her to focus on the decisions that are critical to the system's success.

Unlike any existing approach [1], [5], [13], GuideArch explicitly represents the inherent uncertainty in the knowledge and incorporates that in the analysis. It enables an incremental method of making and refining architectural decisions throughout the engineering process. As the rough estimates in the early stages give way to precise estimates in the later stages, GuideArch allows the architect to refine the models and explore other suitable alternatives.

GuideArch employs *fuzzy mathematical* methods [24] to reason about uncertainty. We have devised a novel fuzzy operator that forms the foundation for quantitatively comparing architectural candidates under uncertainty. The fuzzy operator is then used to develop advanced analysis techniques, including optimization and ranking of architectures, and identification of critical design decisions.

The remainder of this paper is organized as follows. Section II describes a case study and uses it to motivate the research. Section III provides an intuitive description of the approach, while Sections IV-VI present the details. Section VII provides a thorough evaluation of the research. Section VIII outlines the related research. The paper concludes with a discussion of limitations and future research.

II. Motivation and Research Objective

We use a mobile software system, called *Situational Awareness System* (*SAS*), to motivate, describe, and evaluate our research. SAS is developed in collaboration with a government

TABLE I
OVERVIEW OF SAS CASE STUDY.

Decisions▼	Alternatives▼
Location Finding	1: GPS
	2: Radio triangulation
File Sharing Package	1: OpenIntents
	2: In house
Report Synchronization	1: Explicit
	2: Implicit
Chat Protocol	1: XMPP (Open Fire)
	2: In house
Map Access	1: On demand (Google)
	2: Cached on Server
	3: Preloaded (ESRI)
Hardware Platform	1: Nexus 1 (HTC)
	2: Droid (Motorola)
Connectivity	1: Wi-Fi
	2: 3G on Nexus 1
	3: 3G on Droid
	4: Bluetooth
Database	1: MySQL
	2: sqlLite
Architectural Pattern	1: Facade
	2: Peer-to-peer
	3: Push-based
Data Exchange format	1: XML
	2: Compressed XML
	3: Unformatted data

Properties▼
Battery Usage
Response Time
Reliability
Ramp up Time
Cost
Development Time
Deployment Time

agency for the deployment of personnel in emergency response scenarios. SAS is intended to allow the emergency crew carrying Android devices to share and obtain an assessment of the situation in real-time (e.g., interactive overlay on maps), and coordinate with one another (e.g., send reports, chat, and share video streams). We seized the development of this software system as an opportunity to perform our study.

In architecting the SAS application, a team, consisting of academics and engineers from the agency, was formed to decide among the early design decisions. TABLE I shows the decisions and alternatives that comprised the SAS project. The requirements posed by the entities within the agency sponsoring the project also called for several areas of concern, which the team derived over several project meetings with the various stakeholders. The concerns are referred to as properties of the architecture/system. Although not depicted in the table, some of the decisions depend on one another. For instance, the choice of *Connectivity* depended on the selected *Hardware Platform*.

Fig. 1 illustrates the relationship between decisions, alternatives, and properties. Each decision consists of several viable alternatives, the selection of which results in different candidate architectures. Each architecture in turn exhibit its own unique properties, e.g., *Response Time* and *Battery Usage* in Fig. 1 are two such properties. Our experiences with SAS and other systems show that precisely predicting the impact of an architectural alternative on the system's properties is extremely difficult, particularly in early phases of engineering. This difficulty is due to uncertainty.

Previous approaches targeted at early architecting [1], [5], [13] have ignored uncertainty, and assumed the impact of early architectural decisions can be quantified precisely. We collectively refer to them as the *traditional approaches* in this paper. A body of literature (e.g., [3], [6], [16]) has investigated the use of probabilistic models for representing uncertainty in the system's architecture, but such approaches are only useful in settings where probabilistic information is available (e.g., at runtime), which is often not the case in the early architecture phase.

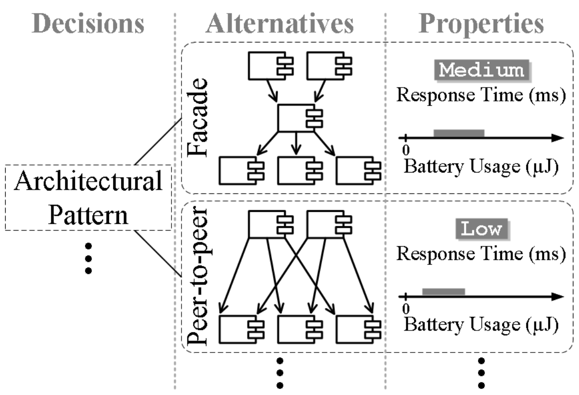


Fig. 1. *Architectural Pattern* decision along with its alternatives and their impact on the properties in SAS's early architecture.

The challenge motivating this research is how to enable the architect make informed decisions in such circumstances? Answering this question requires us to first understand how the practitioners make these decisions. Although an architect may not be able to precisely quantify the impact of decisions on the system's properties, surely she must have some intuition or rough estimate of their impact, as no successful system comes to existence through random decisions [20]. This intuition may be based on the data available from similar designs in other systems, architect's prior knowledge, manufacturer specification, scientific publication, expert judgment, etc. For instance, as depicted in Fig. 1, when presented with two alternative architectural patterns for the system, the architect may be able to distinguish between the two by classifying one's impact on *Response Time* to be *Low*, and the other one as *Medium*. In some other cases, the architect may have more specific knowledge of the situation and specify the impact as a range of values. In the example of Fig. 1, the architect specifies the impact on *Battery Usage* as a range. This paper presents a novel quantitative framework and supporting tool that given such loosely defined estimates help the architect with making the best decisions.

III. APPROACH

In this section, we first define the scope of our research, followed by an intuitive description of our approach.

A. Definition and Scope of Uncertainty

Before delving into the approach, it is important to clarify what we mean by uncertainty. The scope of uncertainty dealt with in our paper has to do with not knowing the exact impact of architectural alternatives on properties of interest, i.e., not being able to precisely specify the impact as a crisp value. However, there are other sources of uncertainty in early architecting that are not tackled in our work. Consider for instance the uncertainty introduced by the following questions:

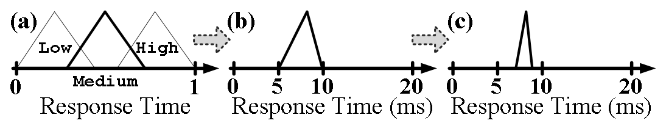


Fig. 2. Evolution of how the architect expresses impact on properties: (a) enumerated values, where the bold one represents the one that is selected, (b) range of concrete values, and (c) convergence towards a crisp value.

Have all of the properties of concern been elicited? Have all of the decisions and alternatives been identified? [19] While the ability to answer such questions is clearly crucial, they fall outside the scope of our research.

B. Representing Uncertainty in Alternatives

As alluded to in Section II, the architect may estimate the impact of an alternative on the properties in two ways: (1) *Range:* When the architect has a rough estimate of the value, the architect could specify it as a range, e.g., a particular choice is anticipated to have $10\mu J$ of battery usage, with $8\mu J$ and $14\mu J$ in optimistic and pessimistic situations, respectively. (2) *Enumeration:* Sometimes the architect may have no way of quantifying the impact, even as a range. In such cases, the architect abstractly specifies the impact of an alternative in relation to other alternatives through enumeration, e.g., certain alternative is likely to have a *Low* response time. Here, *"Low"* may not say much about the actual values (e.g., seconds), but nevertheless carries useful information that influences the decisions early on. These two mechanisms are aligned with the way humans in general conceptualize uncertainty and provide an intuitive method of modeling the architect's imperfect knowledge.

The key contribution of GuideArch is the ability to provide quantitative analysis of the trade-offs given such loose specifications. We achieve this by representing the uncertainty as a *fuzzy value*. A fuzzy value is founded on the concept of *fuzzy set* [24]. In a fuzzy set, the elements have a degree of membership. Degree of membership, also known as *possibility*, is a value between zero and one: a value of zero indicates the element is certainly not a member of the set, a value of one indicates it is certainly a member, and a value in between indicates the extent of certainty it is a member. Fuzzy math is grounded in *possibility theory* [22], which provides an alternative to that of probability theory, and has shown to be particularly useful in settings where uncertainty is due to ambiguity, rather than variability. A common misconception is that fuzzy math is imprecise. On the contrary, fuzzy math, just like probability, provides a precise and sound method of dealing with uncertainty.

We take slightly different approaches to transform the enumerated values and range of values to the corresponding fuzzy representations. As shown in Fig. 2a, the enumerated value is simply mapped to fuzzy values that divide the normalized domain equally. We do this because the architect has no way of quantifying the impact of alternatives on the properties, yet has some intuitions as to how alternatives compare to one another. On the other hand, when the architect can specify the impact as a range (Fig. 2b), we assign the possibility of 1 to the *anticipated* (most likely) value, and possibility of 0 to the *optimistic* and *pessimistic*, respectively. We let the possibility to decrease linearly from the anticipated to the optimistic and pessimistic points.

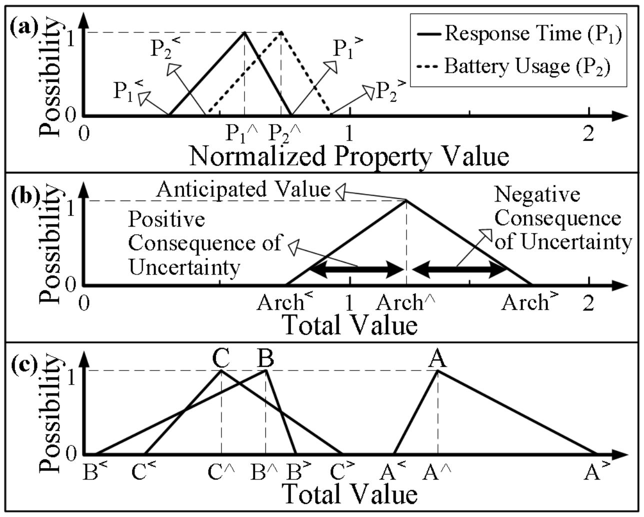


Fig. 3. Uncertainty modeled as fuzzy values using possibility theory: (a) the fuzzy values for *Response Time* and *Battery usage*; (b) their summation to determine the architecture's total value; and (c) the total value for three hypothetical architectures.

As a software project progresses, the architectural models become more concrete and enriched with information collected from simulations and prototypes, subsequently increasing the accuracy with which impact of alternatives can be estimated. Thus, as depicted in Fig. 2, we expect to see the impact of alternatives expressed mostly through enumeration at the outset of a project to give way to ranges of values in later iterations, and eventually converge to crisp values toward the end.

C. Calculating Uncertainty in a Candidate

Using the above approach, the impact of any given alternative on each property is modeled as a *triangular fuzzy value* [24]. As we detail in Section IV-B, these low level measures of uncertainty are combined to calculate the uncertainty in the properties of candidate architectures. For instance, Fig. 3a depicts the fuzzy values corresponding to the *Response Time* (P_1) and *Battery Usage* (P_2) for an architectural candidate. We use "$\wedge$", "$<$", "$>$" to represent anticipated, optimistic, and pessimistic, respectively. Due to uncertainty, the actual value of the property may be anywhere in that range.

Given the fuzzy impact of candidates on properties, we can quantify the overall value of a given architecture. Assuming the properties have been normalized to a value in [0,1] range, and that all properties have the same importance, the total value for an architecture $Arch$ can be calculated as *fuzzy summation* [12] of the impact of candidates on the properties. When fuzzy numbers are summed up, the pessimistic, anticipated, and optimistic values are added independently of each other to arrive at a new fuzzy value. For instance, adding fuzzy values for *Response Time* and *Battery Usage* in Fig. 3a results in the fuzzy value shown in Fig. 3b, which represents the total value of the corresponding architecture $Arch$.[1] Since an architecture with a lower value is preferred, we call the situation in which the actual value is between anticipated and pessimistic the negative consequence of uncertainty (risk), and the situation in which the actual value is between anticipated and optimistic the positive consequence of uncertainty (opportunity).

D. Comparing Candidate Architectures

Fuzzy summation allows us to transform the multi-dimensional problem into a single scalar value, where the architectures can be compared with one another. However, since the scalar value itself is fuzzy, comparing the architectural solutions remains a challenge. When comparing two fuzzy numbers, the one with the "better" range is superior. We say the fuzzy value of one architecture is better than another if it has a: (C1) smaller anticipated value, (C2) larger positive consequence of uncertainty, and (C3) smaller negative consequence of uncertainty [10].

Fig. 3c shows the total value of the properties for three hypothetical architectures (*A*, *B*, and *C*), which are represented as fuzzy values. Using Fig. 3c we describe two possible scenarios that may occur in comparing architectures this way. The first scenario occurs when a given architecture is inferior to others with respect to all three criteria. For instance, in Fig. 3c, architecture *A* is inferior to architectures *B* and *C* with respect to all three criteria. The second scenario occurs when there are trade-offs. For instance, architectures *B* and *C* present a trade-off, as architecture *B* is superior to architecture *C* with respect to C2 and C3, and inferior with respect to C1. Section V describes in detail how we can resolve such trade-offs. The ability to compare architectures under uncertainty provides the basis for exploration and making architectural decisions.

IV. Early Architecture Selection Problem

Equipped with a basic understanding of the problem and approach from the previous section, we now delve into the details. Here, we formally specify the problem of making early architectural decisions under uncertainty.

A. Decisions and Alternatives

We denote the set of architectural *decisions* as set D. Each decision $d \in D$ has several *alternatives*, which we denote as set A_d. We define the set of all alternatives as follows: $A = \bigcup_{d \in D} A_d$. The *architecture space* is a proper subset of the alternatives, where for each decision there exist one and only one alternative that is selected as follows:

$$\begin{aligned} AS \equiv \{ & arch \subseteq A | \forall d \in D : (\exists a \in A_d : a \in arch) \wedge \\ & (\forall a \in arch, a \in A_d : \nexists b \in A_d : b \neq a \wedge b \in arch)\} \end{aligned}$$

Thus, the size of the architecture space is: $\prod_{\forall d \in D} |A_d|$. For example, in our case study shown in TABLE I, we have a total of $2^6 \times 3^3 \times 4 = 6,912$ possible architectural candidates.

[1]The fuzzy summation used here is defined in [12], which has proven to produce another proper possibility distribution.

For each design alternative $a \in A$, we introduce a binary decision variable x_a, which takes the value of 1 if the alternative is selected, and 0 otherwise: $x_{a_i} = 1 \Leftrightarrow a_i \in arch$.

B. Properties and Coefficients

We denote the properties stakeholders are interested in as set P. For alternative $a \in A$ and property $p \in P$ we use $\tilde{c}_{p,a}$ to denote the effect of design alternative a on property p and we call it a *coefficient*. The tilde accent "$\sim$" indicates that the coefficient is a fuzzy value. The set of properties P is partitioned into two subsets P_{min} and P_{max}, indicating properties that need to be minimized (e.g., battery usage) and maximized (e.g., reliability), respectively.

$\tilde{s}_p(arch)$ defines the estimate of property $p \in P$ for a given architecture $arch \in AS$. This function needs to combine the impact of individual alternatives on p to estimate the impact of the candidate architecture on p as a whole. The way this can occur depends on the nature of the property and the structure of the candidate architecture.

Certain properties, such as the total battery usage ($bu \in P$) of a candidate architecture, can be quantified by summing the coefficients (impact) of the selected alternatives as follows: $\tilde{s}_{bu}(arch) = \sum_{i=1}^{|A|} (\tilde{c}_{bu,a_i} \times x_{a_i})$. But for some other properties (e.g., availability), simply summing the contributions of individual alternatives does not result in a meaningful quantification of the corresponding candidate's property. Detailed treatment of such differences is beyond the scope of this paper. We have relied on the previous work [17] that has developed mathematical templates for quantifying several commonly encountered QoS properties at the architectural level. We simply use our fuzzy operators in the context of their templates to deal with uncertainty.

As detailed later in the paper, we would like to reason about the impact of alternatives on several properties with different units/scales, and thus we normalize $\tilde{s}_p$ as follows:

$$\tilde{ns}_p(arch) = (\tilde{s}_p(arch) - min_p)/(max_p - min_p)$$

In cases where the absolute minimum and maximum values are known (e.g., reliability, where minimum is 0% and maximum is 100%), they could be simply used. Otherwise, first, for each $d \in D$, $p \in P$, we let $max_{p,d}$ be equal to the most optimistic/pessimistic value of the alternative $a \in A_d$ that achieves the maximum value for p; next, max_p is calculated by summing all $max_{p,d}$ values. min_p can be calculated in an inverse of the way max_p is calculated.

C. Priorities

Stakeholders are typically concerned about some properties more than others. Identifying their concerns and prioritizing them in terms of risk and importance is the centerpiece of modern software engineering processes [21]. To that end, for each property $p \in P$, we define an integer $\pi_p \in [0, 10]$ indicating the *priority* of property p to stakeholders. The higher the priority, the more important that property is to the stakeholders. We chose this particular representation of priority to be consistent with the existing literature [1], [21], which gives us some confidence that stakeholders can indeed priorities their concerns in this fashion.

D. Total Value of a Candidate Architecture

We let $\tilde{s}(arch)$ represent the total value of a candidate architecture $arch \in AS$, which is calculated by subtracting the total value of the properties that need to be maximized from those that need to be minimized as follows:

$$\tilde{s}(arch) = \sum_{p \in P_{min}} \pi_p \times \tilde{ns}_p(arch) - \sum_{p \in P_{max}} \pi_p \times \tilde{ns}_p(arch)$$

The goal is to find an architecture $arch \in AS$ with the lowest value of $\tilde{s}(arch)$, as it results in minimizing the P_{min} and maximizing the P_{max}. Here contribution of each property is controlled by its priority π_p. Since fuzzy math is closed under these operations, the total value is also a triangular fuzzy number.

E. Dependencies, Conflicts, and Constraints

Some architectural candidates may not be valid. An alternative from one decision may depend on alternative(s) from other decisions, requiring them to be enabled. For instance, as mentioned in Section II, in the case of SAS, *Connectivity* decision depends on the choice of *Hardware Platform* (e.g., since GPS capability is not available on certain devices). We model these dependencies using the function $Dep : A \rightarrow 2^A$, which given an alternative returns a set of alternatives that are dependent on it. The dependency constraint for $arch \in AS$ is then formally specified as follows:

$$\forall a \in arch : x_a \leq \prod_{\delta \in Dep(a)} x_\delta \tag{1}$$

An alternative may also conflict with alternative(s) from other decisions, requiring them to be disabled first, and vice versa. We model these conflicts using the function $Con : A \rightarrow 2^A$, which given an alternative returns a set of alternatives that conflict with it. We formalize these constraints for $arch \in AS$ as follows:

$$\forall a \in arch : x_a \times \sum_{\delta \in Con(a)} x_\delta = 0 \tag{2}$$

A property may have certain thresholds (i.e., limitations). For instance, battery usage may be required to be less than 100 μJ. Given the set Thd representing those property constraints, we formally specify these relationships as follows:

$$\begin{aligned} \forall p \in P_{max} : Thd_p \leq \tilde{s}_p(arch) \\ \forall p \in P_{min} : \tilde{s}_p(arch) \leq Thd_p \end{aligned} \tag{3}$$

V. Comparing Architectures

Recall from Section IV-D that a smaller value of $\tilde{s}$ indicates a better architecture. We say between two valid architectures $arch_1, arch_2 \in AS$, $arch_1$ is better than $arch_2$, if:

$$\tilde{s}(arch_1) \leq \tilde{s}(arch_2) \tag{4}$$

This is a fuzzy comparison [7], [10], since the two sides are fuzzy numbers. Here we are comparing the fuzzy range of possible values for the two architectures. We formalize this by breaking down the fuzzy operator into three comparisons. Let $z_a \equiv S^\wedge$ represent the anticipated value, $z_p \equiv |S^\wedge - S^<|$ represent the positive consequence of uncertainty, and $z_n \equiv |S^> - S^\wedge|$ represent the negative consequence of uncertainty. Fig. 4 provides the intuition behind the three comparisons, where a smaller value of z_a and z_n, and a larger value of z_p are collectively considered to be representative of a smaller fuzzy value, and thus a better architecture. We thus rewrite Equation 4 as:

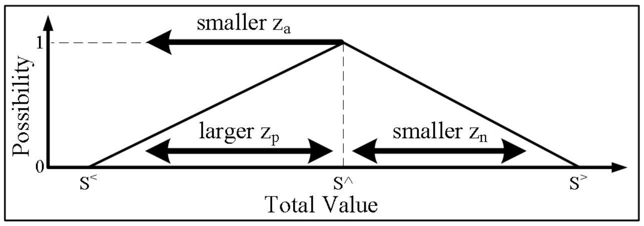


Fig. 4. Intuition behind fuzzy comparison operator.

$$\tilde{s}(arch_1) \leq \tilde{s}(arch_2) \Leftrightarrow z_{a_1} \leq z_{a_2}, z_{p_1} \geq z_{p_2}, z_{n_1} \leq z_{n_2} \tag{5}$$

The three comparisons on the right are formal representations of the three criteria from Section III-D.

Our fuzzy comparison operator is an instance of a multi-dimensional comparison [4], and to decide which range is lower, we first need to transform it to an equivalent single-dimensional comparison. This is necessary to allow us to reason about the trade-offs, such as those depicted in Fig. 3c. The transformation process entails (1) normalizing the values being compared, (2) combining the comparison dimensions, and (3) if necessary, weighting the comparisons differently. In the remainder of this section, we provide a detailed description of these three steps.

A. Normalizing the Values Being Compared

Since the z values are defined differently in terms of $\tilde{s}(arch)$, they are not comparable. Therefore, before combining the comparisons, we first have to normalize the z values. We use normalizing linear membership function [14], which is a function μ that maps each z to a value between 0 and 1: $\forall i \in \{a, p, n\} : (\mu_{z_i} : dom(z_i) \longrightarrow [0, 1])$.

This allows us to have z values with the same range. For defining each function μ, we first need to determine the two extremums for each z: the extremum minimizing z is called *Positive Ideal Solution* (PIS), and the one maximizing z is called *Negative Ideal Solution* (NIS). We can obtain these values by performing the following optimizations, where $argmin$ and $argmax$ find the minimum and maximum values respectively:

$$\begin{aligned} z_a^{PIS} &\equiv argmin_{(arch \in AS)} z_a, z_a^{NIS} \equiv argmax_{(arch \in AS)} z_a \\ z_p^{PIS} &\equiv argmax_{(arch \in AS)} z_p, z_p^{NIS} \equiv argmin_{(arch \in AS)} z_p \\ z_n^{PIS} &\equiv argmin_{(arch \in AS)} z_n, z_n^{NIS} \equiv argmax_{(arch \in AS)} z_n \end{aligned}$$

Note that the NIS and PIS definitions for z_a and z_n are reverse of that of z_p due to their semantic differences (i.e., we prefer a solution with small z_a and z_n, and large z_p). We specify μ to return 0 for the PIS value, 1 for the NIS value, and proportionally linear between the two extremums:

$$\mu_{z_a} \begin{cases} 0 & z_a < z_a^{PIS} \\ \frac{z_a - z_a^{PIS}}{z_a^{NIS} - z_a^{PIS}} & \begin{array}{l} z_a^{PIS} \leq z_a \\ z_a \leq z_a^{NIS} \end{array} \\ 1 & z_a^{NIS} < z_a \end{cases} \mu_{z_p} \begin{cases} 1 & z_p < z_p^{NIS} \\ \frac{z_p^{PIS} - z_p}{z_p^{PIS} - z_p^{NIS}} & \begin{array}{l} z_p^{NIS} \leq z_p \\ z_p \leq z_p^{PIS} \end{array} \\ 0 & z_p^{PIS} < z_p \end{cases}$$

μ_{z_n} is specified similar to μ_{z_a}. Fig. 5 shows instances of μ_{z_a} and μ_{z_p} that normalize the outputs of z_a and z_p, respectively. Since the definitions of NIS and PIS are reversed, the normalizing function μ_{z_a} and μ_{z_n} are increasing, while μ_{z_p} is decreasing.

Defining the normalization functions this way also allows us to flip the comparison for z_p. In other words, $\mu_{z_{p_1}} \leq \mu_{z_{p_2}}$ becomes the normalized equivalent of $z_{p_1} \geq z_{p_2}$. Thus, we rewrite Equation 5 using the normalized values as follows:

$$\tilde{s}(arch_1) \leq \tilde{s}(arch_2) \Leftrightarrow \forall i \in \{a, p, n\} : \mu_{z_{i_1}} \leq \mu_{z_{i_2}} \qquad (6)$$

B. Combining the Comparisons

Given that now we are dealing with three comparisons that have the same range and direction (i.e., less than or equal), we use a conventional technique [14], [23] to transform the multi-dimensional comparison of Equation 6 to a single-dimensional form. We define φ to be the maximum of the three normalized values as follows:

$$\varphi = max_{i \in \{a,p,n\}} \mu_{z_i} \qquad (7)$$

Therefore, $\tilde{s}(arch_1) \leq \tilde{s}(arch_2) \Leftrightarrow \varphi_1 \leq \varphi_2$. While this transformation may not hold for three arbitrary comparisons, since here the numbers are constrained by the triangular relationship that comprises a fuzzy value, the transformation is universally valid (see [23]). Hence, if the value of φ_1 for $arch_1 \in AS$ is less than the value of φ_2 for $arch_2 \in AS$, we conclude $arch_1$ has a lower range, and more desirable. For all practical purposes, it is also possible to use the summation of the normalized values as a collective approximation of the extent in which one architecture is better than the other [10].

C. Weighting the Comparisons

In the above formulation, the three comparisons have the same importance, which is not necessarily the case in certain domains. For instance, in risk-averse domains, it is typically desirable to put more emphasis on reducing the negative consequence of uncertainty (risk) and achieve some level of assurance. Thus, a conservative architecture, where minimizing z_n takes precedence over others, is preferable. We achieve this by assigning weights w_a, w_p, w_n, where $w_a + w_p + w_n = 1$, to normalized values μ_{z_a}, μ_{z_p}, and μ_{z_n}, respectively. The weights specify the importance of each comparison relative to

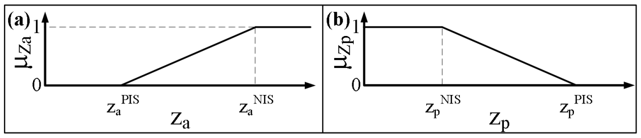


Fig. 5. Linear Membership Function for (a) z_a and (b) z_p.

others. Thus, we rewrite Equation 7 by including the weights as follows:

$$\varphi = max_{i \in \{a,p,n\}} w_i \times \mu_{z_i}$$

VI. Exploring the Solution Space

The ability to compare architectures under uncertainty provides the foundation for architectural exploration. We now describe four ways in which GuideArch helps with this.

A. Identifying Valid Architectures and Critical Constraints

GuideArch allows us to identify the subset of architectural solutions that are "valid", even when there is uncertainty in the knowledge. An architecture is valid, if it satisfies the constraints presented in Section IV-E. An issue is how to check the property constraints, since they involve comparing the crisp value (i.e., Thd) with a fuzzy value (i.e., $\tilde{s}_p(arch)$). Consider for instance a constraint for the cost of realizing an architecture (which is one of the properties as you may recall Table I) to be less than \$10,000. Evaluation of this constraint is challenged by the fact that the estimated cost of realizing the architecture is a fuzzy value.

Constraints, including those specified on properties, are intended to be treated as absolute limitations, and thus we consider the worst case, which occurs at the pessimistic point (i.e., $s_p^>$). Thus, we rewrite the property constraints of Equation 3 as follows:

$$\begin{array}{c} \forall p \in P_{max} : Thd_p \leq s_p^>(arch) \\ \forall p \in P_{min} : s_p^>(arch) \leq Thd_p \end{array} \qquad (8)$$

Comparing the threshold with the worst case ensures an architectural candidate is valid, even when the negative consequence of uncertainty takes effect.

The ability to identify valid architectures not only ensures the architect does not pick a solution that is invalid, but can also be used to provide useful statistical measures. GuideArch provides the percentage of architectures in the solution space that are disqualified by each constraint. This allows the architect to identify the limiting constraints, and to explore their trade-offs. If certain trade-offs are deemed appropriate, they could be communicated to other stakeholders, and thereby set the stage for revising the constraints through negotiations.

B. Finding the Optimal Architecture

GuideArch could also be used to find the *optimal* architecture under uncertainty. An architecture is optimal for a given problem if it achieves the minimum value of φ and satisfies the three types of constraints described in Section IV-E. We define this as a linear programming problem of $argmin_{(arch \in AS)} \varphi_{arch}$, which is subject to the constraints of Equations 1, 2, and 8

Here, the solver uses the approach described in Section V to compare the total value of properties between the candidate solutions. The architecture with the minimum value of φ is the one that achieves the best combination of small anticipated value, small risk, and large opportunity.

C. Ranking the Architectures

The ability to find the optimal solution is complemented with the ability to see a ranking of candidates. There are several reasons for this. First of all, architects bring valuable domain expertise and experience that cannot be represented in existing tools, including GuideArch. Thus, it is possible that an architect may select an architecture that is slightly worse than optimal for reasons that are not modeled in the tool. Reasons for such decisions could range from unfounded biases (e.g., preference for not purchasing products from a particular company) to technical intuitions (e.g., emerging standards). Secondly, the ability to see the top ranked candidates allows the architect to gain insights into why GuideArch selected a solution as optimal. Such rankings could help the architect gain a better understanding of the trade-offs, and increase her confidence in the analysis.

GuideArch uses the ability to compare architectures, described in Section V, to also rank them from best (top) to worst (bottom). We let R represent the ranking of top t architectures as an ordered list:

$$R \equiv \langle arch_1, \ldots, arch_t \rangle \text{ where } arch_i \in AS \wedge$$
$$(\forall arch_i \in R : \nexists c \in AS \wedge c \notin R : \tilde{s}(c) \leq \tilde{s}(arch_i)) \wedge$$
$$(\forall arch_i, arch_k \in R, i \leq k : \tilde{s}(arch_i) \leq \tilde{s}(arch_k))$$

Value of t is a configurable threshold in GuideArch. Here, candidates with better (lower) range appear at the top.

D. Identifying and Ranking the Critical Decisions

Modern software processes advocate an iterative approach, where in each iteration the decisions made in the previous cycles are assessed and risks are mitigated [21]. Generally, it is desirable to resolve decisions that pose a high risk early on, and architecture is typically considered to provide the appropriate level of abstraction to enable such analysis [21].

GuideArch helps the architect identify decisions that are likely to be critical to the success of the project, and thus pose the greatest risk. The insight is that a decision d is likely to be crucial if it satisfies the following two criteria:

1) *Magnitude of Impact*: The selected alternatives from a decision have a big impact on the properties of top ranked architectures. This is reasonable, as the architect is likely to select one of the top ranked architectures, and if a particular decision contributes heavily to those architectures, it is likely to be a crucial decision, i.e., getting it "wrong" results in a large error.
2) *Uncertainty of Impact*: The impact of selected alternatives from decision d on the top ranked architectures is highly uncertain. When there is a lot of uncertainty in the impact of a decision on the top ranked architectures, there is an increased possibility of error.

When the two criteria are present in a decision, we say that the decision is critical, as it is both significant and risky.

We quantify the impact of an alternative $a \in A$ on the properties as follows:

$$\tilde{E}_a = \sum_{p \in P_{min}} \left(\pi_p \frac{\tilde{c}_{p,a}}{max_p} \right) - \sum_{p \in P_{max}} \left(\pi_p \frac{\tilde{c}_{p,a}}{max_p} \right)$$

We then define the impact of a decision $d \in D$ on the properties of the top t ranked architectures as follows:

$$\tilde{E}_d = \sum_{r=1}^{t} \tilde{E}_a \text{ where } a \in arch_r \wedge a \in A_d$$

$\tilde{E}_d$ is the magnitude of impact due to decision d on the top ranked architectures. This approach gives equal weight to the choices made in the ranked architectures. However, since the architect is more likely to select from architectures that are ranked at the top, over those that are ranked at the bottom, we discount the influence based on the order. Thus, we reformulate the above equation by incorporating a *logarithmic decay* to discount the influence of alternatives as we traverse from the top to bottom of ranked architectures: $\tilde{E}_d = \sum_{r=1}^{t} \tilde{E}_a \times e^{-\gamma r}$ where $e^{-\gamma r}$ is the formulation of logarithmic decay, and γ is the decay factor. The larger γ, the more emphasis is placed on the alternatives appearing at the top of the ranking. This is a heuristic that per our experience most naturally captures the increased likelihood of the architect selecting the higher ranked architectures, but using other decay factors (e.g., *linear decay*) is also possible.

Unlike the ranking of architectural candidates, where we are interested in the best solutions, here we are interested in finding the most critical (worst) decisions. We use the fuzzy comparison operator to rank the decisions from worst (top) to best (bottom). Recall from Section V that this operator not only measures the magnitude of impact, but also the range of uncertainty. Let L be the ranking of n most critical decisions as an ordered list:

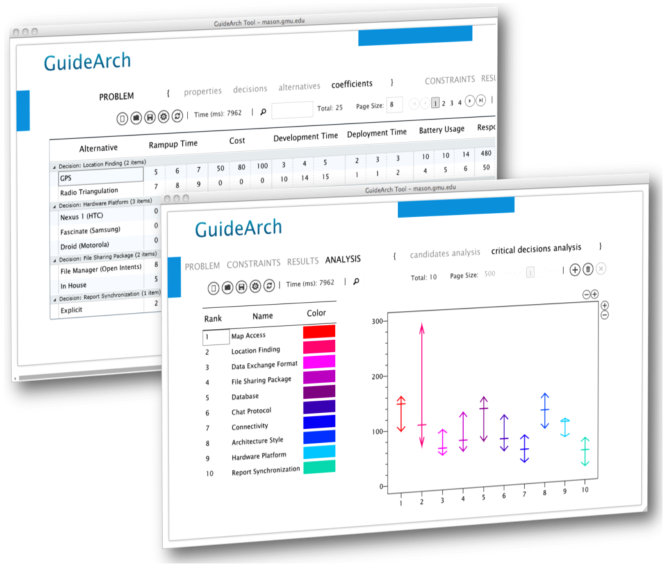
$$L \equiv \langle d_1, \ldots, d_n \rangle \text{ where } d_i \in D \wedge$$
$$(\forall d_i \in L : \nexists c \in D \wedge c \notin L : \tilde{E}_c \geq \tilde{E}_{d_i}) \wedge$$
$$(\forall d_i, d_k \in L, i \leq k : \tilde{E}_{d_i} \geq \tilde{E}_{d_k})$$

The top ranked decisions are good candidates to be investigated further to mitigate risk. Based on this information, the architect may take a number of actions, such as expanding the critical decisions by allowing for additional alternatives, or simply reducing the uncertainty by investing resources and time (e.g., prototyping) to develop a better understanding of the alternatives' impact.

VII. Evaluation

Evaluation of software architecture research is difficult, as it often hinges on industry participation. We feel fortunate to have had a unique opportunity to employ and evaluate our research in the context of a real world project. GuideArch was used by a team of engineers and academics to explore the architectural space of the SAS project. In addition, we have tried to augment our experiences in the real-world with the experiments performed in the laboratory.

The study was supported by an interactive tool realizing GuideArch. Fig. 6 shows two snapshots of the tool: in the back we have the screen used for getting the architect's input as to the expected range of impact each alternative may have on the system properties (recall $\tilde{c}_{p,a}$), while in the front we have the screen displaying the critical decisions and their impact (recall $\tilde{E}_d$) on the properties of the top ranked architectures. The tool is web-accessible to facilitate sharing,


GuideArch Tool – mason.gmu.edu
GuideArch
PROBLEM
{ properties decisions alternatives coefficients }
CONSTRAINTS RESU
Time (ms): 7962
Total: 25
Page Size: 8
1 2 3 4
GuideArch Tool – mason.gmu.edu
GuideArch
PROBLEM CONSTRAINTS RESULTS ANALYSIS
{ candidates analysis critical decisions analysis }
Time (ms): 7962
Total: 10
Page Size: 500
300
200
100
0
1
2
3
4
5
6
7
8
9
10

Alternative	Rampup Time			Cost			Development Time			Deployment Time			Battery Usage			Respo
Decision: Location Finding (2 items)																
GPS	5	6	7	50	80	100	3	4	5	2	3	3	10	10	14	480
Radio Triangulation	7	8	9	0	0	0	10	14	15	1	1	2	4	5	6	50
Decision: Hardware Platform (3 items)																
Nexus 1 (HTC)	0															
Fascinate (Samsung)	0															
Droid (Motorola)	0															
Decision: File Sharing Package (2 items)																
File Manager (Open Intents)	8															
In House	5															
Decision: Report Synchronization (1 item)																
Explicit	2															

Rank	Name	Color
1	Map Access	🟥
2	Location Finding	🟥
3	Data Exchange Format	🟪
4	File Sharing Package	🟪
5	Database	🟪
6	Chat Protocol	🟪
7	Connectivity	🟦
8	Architecture Style	🟦
9	Hardware Platform	🟦
10	Report Synchronization	🟩

Fig. 6. Snapshots of GuideArch in action showing the coefficients (back) and critical decisions (front).

collaboration, and negotiation among the various stakeholders. We used Microsoft®Silverlight™as our platform, which has appropriate plug-ins for mainstream web browsers. The tool is integrated with Microsoft®Solver Foundation for optimization purposes. The tool also provides several features to enable exploration of the architectural space, including color coding to indicate the importance of decisions and measure the quality of candidate architectures, and ranking of the architectures based on various tunable parameters. For the sake of readability, we do not use snapshots of the tool for presenting the evaluation results, and refer the reader to the project site to access the tool and case study data.[2]

In this section, we first describe some of the salient outcomes of this study, followed by the results obtained through additional experiments in the laboratory.

A. Critical Constraints

One of the early requirements posed by the client in SAS was to keep the cost of a single handheld device (includes the cost of the hardware and off-the-shelf software packages) below \$750. While the team already had a hunch that the requirement was overly constraining, there was no method of establishing the extent of it. In particular, since the impact of most alternatives on properties was uncertain, the team had no way of knowing exactly what portion of the architectural space would be disqualified by such a constraint. Using the technique described in Section VI-A, GuideArch showed that this constraint alone disqualifies 5,040 candidates out of 6,912 potential solutions. This allowed the stakeholders to obtain a quantitative, yet intuitive, assessment of the cost constraint on the choices. In a series of negotiations the stakeholders agreed to relax the constraint by increasing the limit to \$1,000. This increased the number of valid architectures, which in turn resulted in finding better candidates. However, even the relaxed constraint disqualified 4,032 architectures and remained as the critical constraint in the project. In the remainder of analyses and experiments reported here, the relaxed cost constraint was used.

[2]Available at http://mason.gmu.edu/~nesfaha2/Projects/GuideArch.

B. Ranking

Fig. 7 shows the (flattened) triangular fuzzy value of 10 sample SAS architectures calculated by GuideArch. The horizontal axis marks the architectures' rankings. As you may recall from Section V, one architecture is better than another if it has a lower z_a and z_n, and a larger z_p. In SAS, we gave equal weights to the satisfaction of each of those conditions (i.e., $w_a = w_n = w_p = 1/3$). Looking at Fig. 7 we can gain insights into the analysis performed by GuideArch. For instance, 1st architecture, which is picked as optimal by GuideArch, has the best combination of three z values, i.e., it has a lower z_a, smaller z_n, and larger z_p than the majority of candidates. As a result, while 1st architecture may be slightly inferior to some candidates with respect to one of the three conditions, it achieves the best set of trade-offs.

C. Optimal Architecture

To illustrate the benefits of GuideArch, we compare it against what we call the traditional approach. Recall from Section II that by traditional approach we collectively refer to any prior research that performs the analysis based on point estimates (i.e., ignores uncertainty). More specifically, here we compare the results of GuideArch against that of ArchDesigner [1], which has aimed to solve a similar problem as that described in this paper, albeit without considering uncertainty.

We observed that the traditional approach would have selected the 486th candidate in Fig. 7 as the optimal solution. Traditional approaches only minimize the anticipated value (z_a). Comparing the difference between 1st and 486th candidates sheds light on the contributions of GuideArch. 486th architecture achieves the lowest z_a among all valid architectures, including the 1st architecture. However, 486th architecture has a very large negative consequence of uncertainty, which has been ignored by the traditional approach. On the other hand, GuideArch selects a solution that has a slightly inferior z_a, but with a better range of uncertainty.

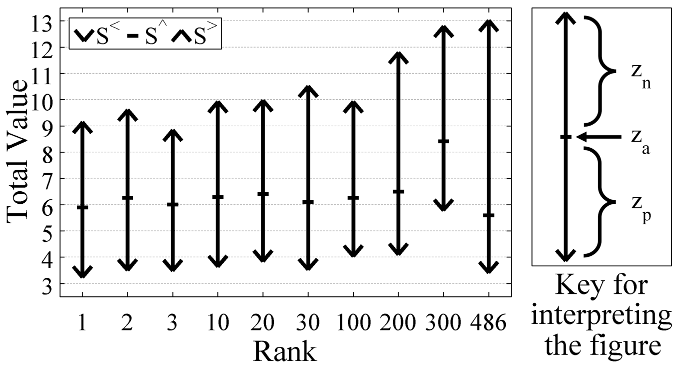


Fig. 7. The triangular fuzzy value of 10 architectural candidates in SAS.

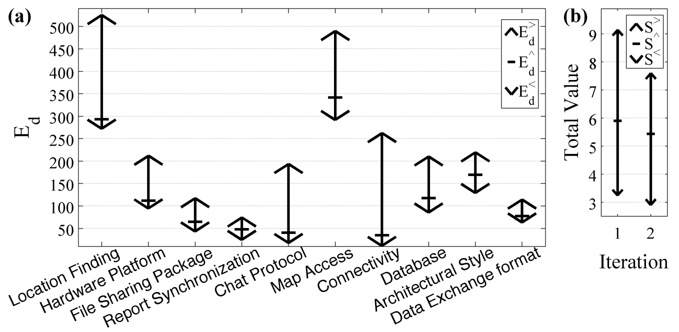


Fig. 8. Critical decisions: (a) comparing the decisions in the first iteration and (b) improvement in the optimal architecture after revising the critical decision in the subsequent iteration.

D. Critical Decisions

GuideArch was also used to identify the critical decisions. Fig. 8a shows the estimated impact of decisions (i.e., $\tilde{E}_d$) as well as the ranges of uncertainty in those estimates, which as you may recall from Section VI-D determine their criticality. We can see *Location Finding* is the most critical decision, since (1) it has a large effect on the ranked architectures, indicated by high y-axis position, and (2) has a very large range of uncertainty, indicated by the span of the arrow. If we use the fuzzy comparison operator (recall Section V), *Location Finding*'s $\tilde{E}_d$ is larger than all other decisions.

This analysis formed the first iteration of using GuideArch. It helped the team identify the critical decisions early on, and focus additional efforts on studying them. As a result, the team came across an advanced state-of-the-art variation of the traditional radio triangulation [15] that presented the project with a new alternative for *Location Finding*. Given that a prototype of this solution had been developed, evaluated, and published [15], the new alternative was estimated to have significantly smaller battery usage, faster response time, and smaller range of uncertainty. GuideArch was applied to the revised problem. Fig. 8b shows the range of total value (i.e., $\tilde{s}$) for the best candidate architecture picked by GuideArch in this second iteration, compared to the best candidate picked in the first iteration. As Fig. 8b shows, the introduction of the new alternative (i.e., smart radio triangulation) improved the total value of the optimal architecture, and reduced the uncertainty.

E. Tuning GuideArch

In the SAS project, we gave the same weight to the three comparisons (i.e., $w_a = w_n = w_p = 1/3$). However, recall from Section V-C that weights could be used to tune GuideArch to be more conservative or bold in its analysis. We performed a set of experiments on the SAS model to assess the impact of weights on the optimal architecture selected by GuideArch.

To allow for comparison, in Fig. 9, we show the result for the balanced weight assignment (i.e., [1/3,1/3,1/3]), which corresponds to the candidate ranked 1st in Fig. 7. Also note that when $w_a = 1$ and $w_n = w_p = 0$, GuideArch behaves exactly like the traditional approach. Therefore, the optimal architecture for that weight assignment in Fig. 9 (i.e., [0,1,0]) is the same as candidate ranked 486th in Fig. 7. This is because the consequence of uncertainty is ignored.

As expected, in the two experiments with high w_n, GuideArch selects a conservative solution, i.e., puts more emphasis on minimizing the negative consequence of uncertainty. In the two experiments with high w_p, GuideArch selects a risky solution, i.e., puts more emphasis on maximizing the positive consequence of uncertainty. Both approaches come at the cost of achieving mediocre anticipated total value. While in our experiments a balanced weight assignment has shown to achieve the most appropriate trade-offs, we can envision situations in which placing emphasis on one of the comparisons may be more appropriate, which GuideArch allows for naturally.

VIII. Related Work

Making architectural decisions is a problem that has been studied from both design-time and run-time perspectives. The uncertainty issues in the latter have been mainly researched in the area of autonomic computing [8]. Here we discuss only those targeted at design-time, since that has been the focus of our work.

ArchDesigner [1] is an approach to find an optimal architecture that meets conflicting stakeholders' quality goals. ArchDesigner uses linear programming to find the optimal architecture. CBAM [13] is a quantitative approach for economic modeling of software engineering decisions, which builds upon ATAM [5]. CBAM provides the cost and benefit of different architectural candidates. ArcheOpterix [2] is a tool for optimizing an embedded system's architecture. It uses evolutionary algorithms for multi-objective optimization of such systems. While many of these approaches acknowledge the challenges posed by uncertainty, none addresses it explicitly and via a mathematical framework.

Palladio [3] uses information about components comprising the architecture to derive analytical models and simulate the system's performance. Random variables are used to specify uncertainty in service demands and iterations. Meedeniya et al. [16] estimate the reliability of a given software architecture by combining reliability of its elements (expressed as probability distributions) using Monte Carlo simulation. These approaches are complementary to GuideArch, as they could be used in the later phases of software architecting, where probabilistic

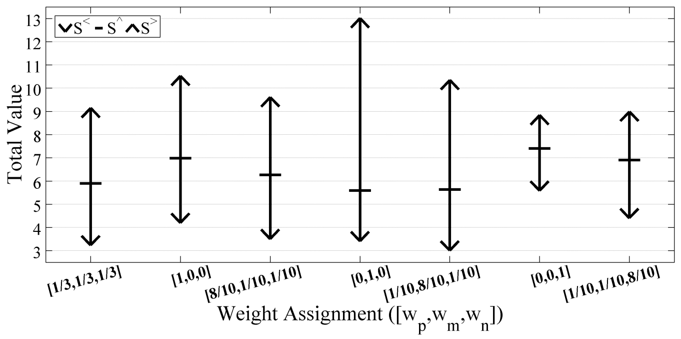


Fig. 9. The optimal architecture for different weights.

information about the components comprising a software system can be obtained (e.g., from prototypes).

Doyle et al. [6] present an approach to compare candidate architectures, assuming the availability of a probability distribution representing the response time of each architecture. Instead of fuzzy math, they rely on extensive integration to compare the distributions, which are computationally expensive, making their approach inapplicable to large systems.

Noppen et al. [18] use *design tree* to navigate in the design space, which may include imperfect information. They consider each alternative independent of others. GuideArch, on the other hand, considers all alternatives influencing the system's properties at the same time.

Finally, in a short position paper [9], we have argued for the utility of fuzzy logic in addressing the uncertainty issues in early architecture.

IX. Conclusion

In any software project, early architectural decisions represent some of the most crucial decisions engineers ever make. Yet there is a lack of techniques and tools for helping the engineers to make those decisions. We presented GuideArch, a novel framework that guides the engineers in making the best choices possible under uncertainty. It provides a combination of capabilities, such as ranking the architectures, finding the optimal, and identifying the critical decisions, that collectively help with the exploration of the solution space. GuideArch is tunable, allowing the engineer to set the analysis to be as conservative as desired.

While thorough evaluation of GuideArch in the context of a case study as well as laboratory experiments allowed us to experience its benefits firsthand, it also revealed several areas of future improvement. One area of future work is to extend the current model from a *unified* stakeholder perspective to a *multiple* stakeholder perspective. We currently assume all stakeholders have agreed on the impact of alternatives on properties and their priorities. However, this may not always be the case, presenting GuideArch with yet another source of uncertainty. Future work also includes extending GuideArch to deal with the other types of uncertainty discussed in Section III-A. Moreover, GuideArch currently represents uncertainty as a *triangular fuzzy value*, which is not only the most widely used fuzzification approach, but also universally applicable. However, in our future work, we also plan to experiment with alternative representations (e.g., *trapezoidal* and *Gaussian*) that seem to offer some unique trade-offs.

Acknowledgements

We would like to thank Thabet Kacem and Ehsan Kouroshfar for their help and the anonymous ICSE reviewers for their feedback. This work is supported in part by awards P010033816 and P010033886 from Science Applications International Corporation, D11AP00282 from the Defense Advanced Research Projects Agency, CCF-1217503 and CCF-1252644 from the National Science Foundation. Any opinions and conclusions expressed in this material are those of the authors and do not reflect the views of the funding agencies.

References

[1] T. Al-Naeem, I. Gorton, M. A. Babar, F. Rabhi, and B. Benatallah, "A quality-driven systematic approach for architecting distributed software applications," in *Int'l Conf. on Software Engineering*, St. Louis, Missouri, May 2005, pp. 244–253.

[2] A. Aleti, S. Bjornander, L. Grunske, and I. Meedeniya, "ArcheOpterix: an extendable tool for architecture optimization of AADL models," in *ICSE Workshop on Model-Based Methodologies for Pervasive and Embedded Software*, Vancouver - Canada, May 2009, pp. 61–71.

[3] S. Becker, H. Koziolek, and R. Reussner, "The palladio component model for model-driven performance prediction," *J. Syst. Softw.*, vol. 82, no. 1, pp. 3–22, Jan. 2009.

[4] S.-J. Chen and C.-L. Hwang, *Fuzzy Multiple Attribute Decision Making: Methods and Applications*, 1st ed. Springer, Feb. 1992.

[5] P. Clements, R. Kazman, and M. Klein, *Evaluating Software Architectures: Methods and Case Studies*. Addison-Wesley Professional, Nov. 2001.

[6] G. S. Doyle, "A methodology for making early comparative architecture performance evaluations," Ph.D. dissertation, George Mason University, Dec. 2010. [Online]. Available: http://hdl.handle.net/1920/6361

[7] D. Dubois and H. Prade, "Ranking fuzzy numbers in the setting of possibility theory," *Information Sciences*, vol. 30, no. 3, pp. 183–224, Sep. 1983.

[8] N. Esfahani, E. Kouroshfar, and S. Malek, "Taming uncertainty in self-adaptive software," in *Int'l Symp. on the Foundations of Software Engineering*, Szeged, Hungary, Sep. 2011, pp. 234–244.

[9] N. Esfahani, K. Razavi, and S. Malek, "Dealing with uncertainty in early software architecture," in *Int'l Symp. on the Foundations of Software Engineering*, Cary, North Carolina, Nov. 2012.

[10] G. Facchinetti and R. Ghiselli Ricci, "A characterization of a general class of ranking functions on triangular fuzzy numbers," *Fuzzy Sets and Systems*, vol. 146, no. 2, pp. 297–312, Sep. 2004.

[11] D. Garlan, "Software engineering in an uncertain world," in *FSE/SDP Wrkshp. on the Future of Software Engineering Research*, Santa Fe, New Mexico, Nov. 2010, pp. 125–128.

[12] A. Kaufmann and M. M. Gupta, *Fuzzy mathematical models in engineering and management science*. North-Holland, 1988.

[13] R. Kazman, J. Asundi, and M. Klein, "Quantifying the costs and benefits of architectural decisions," in *Int'l Conf on Software Engineering*, Toronto, Canada, May 2001, pp. 297–306.

[14] Y.-J. Lai and C.-L. Hwang, "A new approach to some possibilistic linear programming problems," *Fuzzy Sets Syst.*, vol. 49, no. 2, pp. 121–133, Jul. 1992.

[15] K. Lin, A. Kansal, D. Lymberopoulos, and F. Zhao, "Energy-accuracy trade-off for continuous mobile device location," in *Int'l Conf. on Mobile systems, applications, and services*, San Francisco, California, Jun. 2010, pp. 285–298.

[16] I. Meedeniya, I. Moser, A. Aleti, and L. Grunske, "Architecture-based reliability evaluation under uncertainty," in *Int'l Conf on the Quality of Software Architectures*, Boulder, CO, Jun. 2011, pp. 85–94.

[17] D. A. Menasce, J. P. Sousa, S. Malek, and H. Gomaa, "QoS architectural patterns for self-architecting software systems," in *Int'l Conf. on Autonomic Computing*, Washington, DC, Jun. 2010, pp. 195–204.

[18] J. Noppen, P. van den Broek, and M. Aksit, "Software development with imperfect information," *Soft Comput.*, vol. 12, no. 1, pp. 3–28, Aug. 2007.

[19] M. Nowak, C. Pautasso, and O. Zimmermann, "Architectural decision modeling with reuse: challenges and opportunities," in *ICSE Wrkshp on Sharing and Reusing Architectural Knowledge*, Cape Town, South Africa, May 2010, pp. 13–20.

[20] D. Tofan, M. Galster, and P. Avgeriou, "Capturing tacit architectural knowledge using the repertory grid technique," in *Int'l Conf on Software Engineering*, Waikiki, Honolulu, Hawaii, May 2011, pp. 916–919.

[21] Y. Yang and B. Boehm, "Improving process decisions in COTS-based development via risk-based prioritization," *Software Process: Improvement and Practice*, vol. 12, no. 5, pp. 449–460, Sep. 2007.

[22] L. A. Zadeh, "Fuzzy sets as a basis for a theory of possibility," *Fuzzy Sets Syst.*, vol. 100, pp. 9–34, Jun. 1999.

[23] H. J. Zimmermann, "Fuzzy programming and linear programming with several objective functions," *Fuzzy Sets and Systems*, vol. 1, no. 1, pp. 45–55, Jan. 1978.

[24] H.-J. Zimmermann, *Fuzzy Set Theory and its Applications (4th Edition)*, 4th ed. Springer, Oct. 2001.