Department of Computer Science
George Mason University
Technical Report Series

4400 University Drive MS#4A5
Fairfax, VA 22030-4444 USA
http://cs.gmu.edu/
703-993-1530

Guided Exploration of the Architectural Solution Space in the Face of Uncertainty

Naeem Esfahani
nesfaha2@gmu.edu

Sam Malek
smalek@gmu.edu

Technical Report GMU-CS-TR-2011-3

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 guiding the exploration of the architectural solution space under uncertainty. It provides techniques and tools that help the engineer to make informed decisions. The approach has been thoroughly evaluated in a project aimed at reengineering a mobile software system.

Keywords

Uncertainty; Software Architecture; Decision Making

1. 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.[1] 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 development process. 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 early architectural decisions is a complex task mired with lots of uncertainty. Getting them "wrong" poses one of the greatest risks 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 [6], CBAM [12], and ArchDesigner [2]. 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 on this front, the existing approaches do not systematically deal with uncertainty [11]. In fact, there is no systematic

[1] While our definition of "early architecture" incorporates decisions dealing with hardware, system, and software, our focus in this paper is mainly on software decisions.

method of even comparing two architectures under uncertainty, let alone making the "right" decisions in such circumstances [2][11].

In this paper, we present a framework aimed at *guiding the uncertainty-driven exploration of architectural space*, in short *GuideArch*. It allows the architect to make informed decisions using imperfect information. It does so by providing a number of key capabilities, including the ability to rank the candidate architectures based on their traits, and identifying the most critical constraints and design decisions. 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 [2, 6, 12], GuideArch explicitly represents the inherent uncertainty of 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 [21] to reason about uncertainty. We have devised a novel fuzzy operator that forms the foundation for quantitative comparison of 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.

Our experience with applying the approach in a project consisting of numerous design choices and constraints has been very positive. GuideArch allowed the stakeholders to gain better insights into the architectural alternatives and avoid risky solutions early in the project.

The remainder of paper is organized as follows. Section 2 describe a case study and uses it to motivate the research. Section 3 provides an intuitive description of our approach, while Sections 046-6 formally present the details. Section 0 provides a thorough evaluation of the research. Section 8 outlines the related research. The paper concludes with a discussion of future work.

2. Motivation

We use a software system, called Emergency Deployment System (EDS) [16], to motivate, describe, and evaluate our research. EDS was previously developed in collaboration with a government agency for the deployment of personnel in emergency response scenarios. EDS consists of two subsystems: *Headquarters* and *Search & Rescue*. The *Headquarters* subsystem manages several *Search & Rescue* subsystems. The *Search & Rescue* subsystem is a mobile application that runs on ruggedized smartphones and tablets, and used by emergency crew members.

The original subsystem was relatively rudimentary, providing support merely for sharing data with *Headquarters*. The recent proliferation of mobile technologies, standards (e.g., Open Handset Alliance [18]), and platforms (e.g., Android [17]) presented an opportunity to improve this part of the system. Therefore, a project for clean slate re-architecting of the subsystem was ensued. The new application was intended to allow the crew to share and obtain an assessment of the situation in real-time (e.g., interactive overlay on maps), coordinate with one another (e.g., send reports, chat, and share video streams), and engage with the *Headquarters* (e.g., receive commands).

We seized the revamping of this software system as an opportunity to perform the following study. In re-architecting the EDS application, a team, consisting of academics and engineers from the agency was formed, to decide among the early design *decisions*, shown on the left most column of TABLE I. Each decision consists of several viable *alternatives,* shown on the second column from the left. 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 shown on the top most row of TABLE I, and referred to as *properties* of the architecture/system. We will revisit TABLE I and its details in Section 3.

Our experiences with EDS 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 observation is also corroborated by other researchers and practitioners [12][2]. As a result, the process of making early architectural choices is a risky proposition mired with uncertainty.

Previous approaches (e.g., [12][2][15]) that support the process of making decisions and optimizing the system's architecture ignore these challenges, which hampers their adoption in real-world risk-averse domains. We collectively refer to these as the *traditional approaches*. We illustrate their shortcoming using a problem in which the objective is to choose from a pool of 16 candidate architectures, such that *Cost* and *Battery Usage* are minimized.

TABLE I. EXPECTED EFFECT OF ARCHITECTURAL ALTERNATIVES ON PROPERTIES, AND PRIORITIES IN *SEARCH & RESCUE* SUBSYSTEM. "<", "^", and ">" COLUMNS INDICATE OPTIMISTIC, ANTICIPATED, AND PESSIMISTIC IMPACT, RESPECTIVELY.

Decisions	Alternatives	Ramp up Time (days)			Cost (dollars)			Development Time (days)			Deployment Time (days)			Battery Usage (µJ)			Response Time (ms)			Reliability (%)		
		<	^	>	<	^	>	<	^	>	<	^	>	<	^	>	<	^	>	<	^	>
Location Finding	**1:** GPS	5	6	7	50	80	100	3	4	5	2	3	3	10	10	14	480	500	990	80	75	60
	2: Radio triangulation	7	8	9	0	0	0	10	14	15	1	1	2	4	5	6	50	100	600	99	92	90
Hardware Platform	**1:** Nexus 1 (HTC)	0	0	0	500	525	530	0	0	0	0	0	0	3	5	5	40	60	65	0	0	0
	2: Droid (Motorola)	0	0	0	520	520	600	0	0	0	0	0	0	4	5	14	50	55	200	0	0	0
File Sharing Package	**1:** File manager (OpenIntents	8	9	10	0	0	0	3	4	6	1	1	2	5	5	6	50	65	70	99	95	90
	2: In house	5	8	12	0	0	0	5	6	15	0	0	0	3	4	7	40	60	100	96	92	80
Report Synchronization	**1:** Explicit	2	2	3	0	0	0	5	6	7	1	2	2	1	3	4	20	30	50	90	88	85
	2: Implicit	1	2	2	0	0	0	3	4	4	1	1	1	7	8	10	1	4	10	99	97	95
Chat Protocol	**1:** Openfire	5	6	7	0	0	0	5	6	8	1	1	2	4	5	6	40	60	70	99	95	94
	2: In house	3	4	14	0	0	0	7	8	20	0	0	0	2	3	12	30	40	200	97	96	60
Map Access	**1:** On demand (Google site)	7	9	10	0	0	0	14	18	21	0	0	0	4	4	12	700	800	900	92	91	70
	2: Cached (Google server)	7	9	10	700	900	950	14	18	21	3	4	5	4	5	12	1	4	500	98	97	85
	3: Preloaded (ESRI)	10	13	14	100	170	200	20	27	30	3	4	5	5	7	7	1	2	3	99	90	85
Connectivity	**1:** Wi-Fi	1	3	4	70	80	85	0	0	0	5	6	7	3	4	5	30	35	40	95	85	80
	2: 3G on Nexus 1	1	2	3	360	400	600	0	0	0	2	3	4	1	2	3	20	25	40	99	88	80
	3: 3G on Droid	1	2	3	360	400	600	0	0	0	2	3	4	2	4	5	20	25	40	99	88	80
	4: Bluetooth	1	2	8	50	70	200	0	0	0	4	5	15	2	3	15	25	30	200	85	85	50
Database	**1:** MySQL	1	2	3	0	0	0	15	17	18	10	15	16	3	6	7	20	25	30	99	90	85
	2: sqlLite	3	4	5	0	0	0	15	16	22	13	14	22	5	5	10	8	10	50	95	90	70
Architectural Style	**1:** Peer-to-Peer	10	11	13	0	0	0	20	26	30	14	18	21	7	8	10	10	20	30	70	66	60
	2: Client-Server	7	8	10	0	0	0	15	16	20	7	9	10	5	6	7	25	30	80	97	95	85
	3: Push-Based	9	10	12	0	0	0	20	24	25	8	9	12	2	4	4	15	25	40	99	94	90
Data Exchange format	**1:** XML	2	3	4	0	0	0	6	7	8	0	0	0	3	4	6	20	35	80	0	0	0
	2: Compressed XML	4	5	6	0	0	0	7	9	10	0	0	0	5	5	7	12	20	35	0	0	0
	3: Unformatted data	1	2	3	0	0	0	3	4	5	0	0	0	1	1	3	3	10	15	0	0	0
Priorities		2			1			2			2			9			7			3		

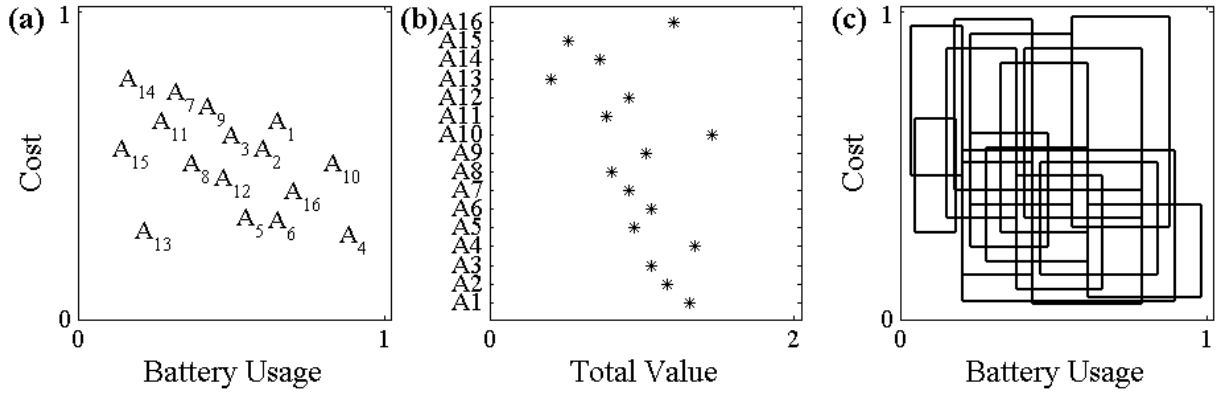

Figure 1. Quantitatively assessing architectural candidates: (a) 16 candidate architecture in a cost vs. battery usage trade-off, (b) simple additive approach to resolve the trade-offs, and (c) cost vs. battery usage under uncertainty, where each rectangle represents the space of values that an architecture may take.

The traditional approaches assume that the architect is able to precisely specify the impact of candidate architectures on properties of interest. If that was the case, then one could visualize the situation as in Figure 1a. Here, for the sake of clarity, the values for *Cost* and *Battery Usage* are normalized between zero and one. Assuming both properties have the same level of importance, to compare the 16 candidates, for each architecture we first sum up the values obtained in the two properties. Figure 1b achieves just that, as it shows the overall value for the candidate architectures. In this space, architectures can be compared with one another. For example, we can see that A_{13} is the best architecture, as it obtains the smallest total value. It is also possible for several architectures to obtain the same value, in which case the architect would need to provide a prioritization scheme, such that more emphasis is placed on certain properties. However, for clarity we do not consider such cases in this section, and revisit that later in the paper. While the aforementioned approach is theoretically sound, it is not useful in practice, as it does not incorporate the underlying uncertainty in the impact of architectural decisions on properties of interest.

The complexity of incorporating uncertainty in the analysis is shown in Figure 1c. Here, the architect's uncertainty is represented in terms of range of impact that an architectural candidate may have on the properties of interest. For example, the impact of a given architecture on *Battery Usage* is no longer a single number, but rather a range of values. As a result, each architectural candidate may obtain a value anywhere within the area occupied by the corresponding rectangle. Clearly, comparing two architectures with overlapping rectangles is difficult. It is not clear how the rectangles in Figure 1c can be transformed to a space where the trade-off analysis can be performed.

To gain a better appreciation for the complexity of this problem consider that the simple example used in Figure 1 consists of only 16 architectural candidates and 2 properties of interest, but a typical software system often consists of many more candidates and properties. For instance, the EDS problem depicted in TABLE I consists of a total of 6,912 potential architectural solutions, each of which could present a trade-off with respect to 7 properties of interest.[2] Clearly, manually exploring such a large space is a big burden. Incorporating uncertainty into the analysis makes a problem that is already challenging so overwhelmingly complex that a manual assessment without the appropriate tools and techniques becomes impossible.

3. Approach

We accept uncertainty as a natural component of architecting a software system, particularly in the early phases of engineering. Our objective is not to eliminate uncertainty, but to provide techniques and tools for making informed decisions in such circumstances. This section provides an intuitive description of our approach; the underlying details and mathematics are then presented in Sections 46-6.

3.1 Representing Uncertainty in Impact of Alternatives

Instead of modeling the anticipated impact of an architectural alternative on the system's properties as a point estimate, we represent it as a *range* of values. Specifying the impact in terms of a range is aligned

[2] In Section 4, we describe how the number of candidate architectures can be calculated.

with the way humans in general conceptualize uncertainty and provides an intuitive method of modeling the architect's knowledge. TABLE I shows these ranges in the EDS system, where for each alternative, the optimistic ("<" column), anticipated ("^" column), and pessimistic (">" column) impact on the properties are provided.

The range of impact may be estimated in a number of ways, including the data available from similar designs in other systems, architect's prior knowledge, prototype of the system, etc. For instance, from prior experience with smartphones, the architect may estimate that *Location Finding* using *GPS* has $10\mu J$ of anticipated battery usage, with $8\mu J$ and $14\mu J$ in optimistic and pessimistic situations, respectively. While there are other elaborate methods of representing uncertainty, such as *probability distribution*, our experience suggests that architects are often not capable of expressing such models. We note, however, that if such models of uncertainty are available, then the range could be easily derived using the techniques described in [9].

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 uncertain parameters as *fuzzy numbers*. A fuzzy number is founded on the concept of *fuzzy set* [20]. In a fuzzy set, the elements have a degree of membership. Degree of membership 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 the element is certainly a member of the set, and a value in between indicates the extent of certainty that the element is a member of the set. Fuzzy math is grounded in *possibility theory* [20], which provides an alternative interpretation of uncertainty to that of probability theory. 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. Fuzzy decision making techniques are also often more efficient than stochastic programming approaches [21]. In addition, since uncertainty in our problem is not due to statistical error or noise, but rather to the imprecision in knowledge, we adopt possibility theory as the foundation of dealing with uncertainty in GuideArch.

We assign the possibility of one to the *anticipated* value, and possibility of zero to the *optimistic* and *pessimistic*, respectively. We use "^", "<", ">" to represent anticipated, optimistic, and pessimistic, respectively. We let the possibility to decrease linearly from the anticipated to the optimistic and pessimistic points. Thus, the effect of each design alternative on each property is modeled as a *triangular fuzzy value* [21]. For instance, Figure 2a depicts the fuzzy values corresponding to the range of *Cost* and *Battery Usage* for an architectural candidate. Due to uncertainty, the actual value of the property may be anywhere in that range.

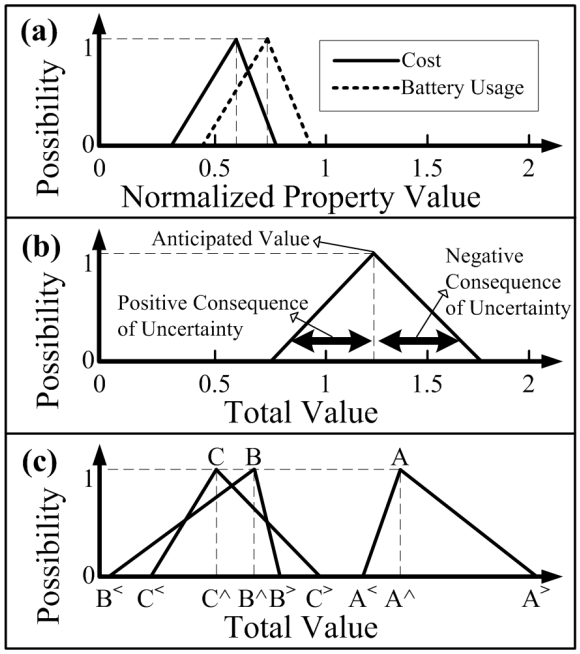

Figure 2. Uncertainty modeled as fuzzy values using possibility theory: (a) the fuzzy values for Cost and Battery usage, (b) their summation to determine the architecture's total value, and (c) the total value for three hypothetical architectures.

3.2 Calculating Uncertainty in a Candidate Architecture

Given the fuzzy impact of alternatives on properties, we can now quantify the overall value of a given architecture. Similar to the approach employed to transform Figure 1a to Figure 1b, we can transform the candidate solutions in Figure 1c to a scalar space, such that they can be compared with one another. The total value for each architecture can be calculated as *fuzzy summation* of the impact of alternatives 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 *Cost* and *Battery Usage* in Figure 2a results in the fuzzy value shown in Figure 2b, which represents the total value of the corresponding architecture. 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, and the situation in which the actual value is between anticipated and optimistic the *positive consequence* of uncertainty.

3.3 Comparing Candidate Architectures under Uncertainty

Fuzzy summation allows us to transform the multi-dimensional problem into a single scalar value, but since the scalar value itself is fuzzy, comparing 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. Figure 2c shows the total value of the properties for three hypothetical architectures (*A*, *B*, and *C*), which are represented as fuzzy values. Using Figure 2c 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 Figure 2c, 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 5 describes in detail how we can resolve such trade-offs. Section 6 describes how the ability to compare architectures under uncertainty provides the basis for exploration of the solution space, including ranking, optimization, and identification of the critical choices.

4. Specification of Architecture Selection Problem

In this section, we formally specify the problem of making early architectural decisions.

4.1 Decisions and Alternatives

We denote the set of architectural *decisions* as set D. For instance, in TABLE I, *Architectural Style* is a decision. Each decision $d \in D$ has several *alternatives*, which we denote as set A_d. For example, the *Architectural Style* decision in TABLE I has three alternatives: *Peer-to-Peer*, *Client-Server*, and *Push-Based*. 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:

$$AS \stackrel{\text{def}}{=} \{arch \subseteq A | (\forall d \in D: \exists a \in A_d: a \in arch) \wedge (\forall a \in arch, a \in A_d: \nexists b \in A_d: b \neq a \wedge b \in arch)\}$$

Thus, the size of the architecture space is: $\prod_{\forall d \in D} |A_d|$ For instance, for EDS problem in TABLE I, we have 6 decisions with 2 alternatives, 3 decisions with 3 alternatives, and one decision with 4 alternatives for a total of $2^6 \times 3^3 \times 4$=6,912 possible architectural candidates.

For each design alternative $a \in A$, we introduce a binary decision variable x_a, which is equal to 1 if the alternative is selected, and 0 otherwise: $x_a = 1 \Leftrightarrow a \in arch$

4.2 Properties and Coefficients

We denote the properties that stakeholders are interested in as set P. In TABLE I properties are shown in the clustered columns (e.g., *Cost*). As described in Section 3, each property is broken down to three values quantifying the uncertainty in the impact of an alternative on that property. 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 "~" indicates that the coefficient is a fuzzy value. The set of properties P is partitioned into two subsets P_{min} and P_{max}, representing properties that need to be minimized (e.g., *Cost*) and maximized (e.g., *Reliability*), respectively.

We assess the contributions of property $p \in P$ to a given architecture $arch \in AS$ by summing the coefficients of the selected alternatives as:

$$\tilde{S}_p(arch) = \sum_{a \in arch} (\tilde{c}_{p,a}\, x_a)$$

The coefficients are included in the summation when the corresponding alternative is selected. Note that since we use fuzzy arithmetic, the result is also a fuzzy number.

As detailed later in this 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:

$$\widetilde{NS}_p(arch) = \tilde{S}_p(arch) / max_p$$

Where max_p is calculated as follows:

$$\forall d \in D, p \in P_{max}: max_{p,d} = maximum_{(a \in A_d)} c^{>}_{p,a}$$

$$\forall d \in D, p \in P_{min}: max_{p,d} = maximum_{(a \in A_d)} c^{<}_{p,a}$$

$$max_p = \sum_{d \in D} max_{p,d}$$

That is, first, for each $d \in D$, $p \in P$, we let $max_{p,d}$ be equal to the 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. Note that max_p needs to be calculated only once for each property. In cases where the absolute maximum value

is known (e.g., reliability, where maximum is 100%), it could be simply used instead.

4.3 Priorities

Stakeholders are typically concerned about some properties more than others. Identifying the stakeholder concerns and prioritizing those in terms of risk and importance is the centerpiece of modern software engineering processes [19]. 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 [2][19], which gives us some confidence that stakeholders can indeed priorities their concerns in this fashion. The last row in TABLE I shows the priorities in our case study.

4.4 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:

$$\tilde{S}(arch) = \sum_{p \in P_{min}} \left(\pi_p \, \widetilde{NS}_p(arch)\right) - \sum_{p \in P_{max}} \left(\pi_p \, \widetilde{NS}_p(arch)\right)$$

Finding an architecture $arch \in AS$ with the lowest value of $\tilde{S}(arch)$ is desirable, since it results in minimizing the P_{min} and maximizing the P_{max} properties. Here contribution of each property is controlled by its priority π_p. Since fuzzy arithmetic is closed under these operations, the total value is also a triangular fuzzy number. From which the range of the total value ($S^<$ and $S^>$) and the anticipated value ($S^\wedge$) can be determined.

4.5 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. An example of this constraint in EDS (see TABLE I) was that the *3G* alternatives for *Connectivity* are dependent on the alternatives for *Hardware Platform*. 3G has different battery consumption estimates depending on the type of hardware platform. We define function $Dep: A \rightarrow \wp(A)$ that given an alternative returns a set of alternatives with dependency relationship to it. The dependency constraint for $arch \in AS$ is then formally specified as follows

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

An alternative may also conflict with alternative(s) from other decisions requiring them to be disabled first, and vice versa. For instance, in EDS (see TABLE I), since *MySQL* could not be installed on smartphones, it had a conflict with the *Peer-to-Peer* alternative, which assumed only peers and no reliance on database connectivity to the backend (*Headquarters*). We define function $Con: A \rightarrow \wp(A)$ that given an alternative returns a set of conflicting alternatives. We formalize these constraints for $arch \in AS$ as follows:

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

A property may have certain thresholds (i.e., limitations). For instance, *Reliability* may be required to be greater than 90% or the *Ramp up Time* [5] may be required to be less than 40 man-days. We use the set Thd to represent those property constraints. We formalize property constraints for $arch \in AS$ as follows:

$$\forall p \in P_{max}: Thd_p \leq \tilde{S}_p(arch)$$
$$\forall p \in P_{min}: \tilde{S}_p(arch) \leq Thd_p$$

4.6 Scope of Our Problem

It is important to note that the scope of uncertainty dealt with in our paper has to do with not knowing the exact impact of alternatives on properties. 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: Have all of the properties of concern been elicited? Have all of the decisions and alternatives been identified? Do the priorities indeed represent the stakeholders' true preferences? While the ability to answer such questions is clearly crucial, they fall outside the scope of this paper.

5. Comparing Architectures under Uncertainty

Recall from Section 4.4 that a smaller value of $\tilde{S}$

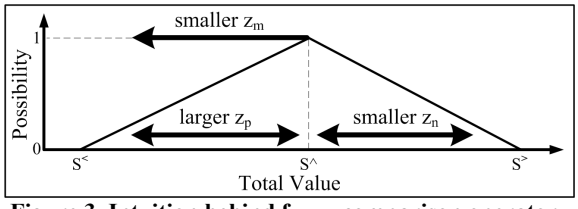

Figure 3. Intuition behind fuzzy comparison operator.

indicates a better architecture. Thus, we say between two valid architectures $arch_1, arch_2 \in AS$, $arch_1$ is better than $arch_2$, if:

$$\tilde{S}(arch_1) \leq \tilde{S}(arch_2) \quad \text{Eq. 1}$$

This is a fuzzy comparison, since the two sides are fuzzy numbers. Here we are comparing the range of possible values for the two architectures. We formalize this by breaking down the fuzzy comparison operator into three concurrent comparisons. Let $z_m \stackrel{\text{def}}{=} S^{\wedge}$ represent the anticipated value, $z_p \stackrel{\text{def}}{=} |S^{\wedge} - S^{<}|$ represent the positive consequence of uncertainty, and $z_n \stackrel{\text{def}}{=} |S^{>} - S^{\wedge}|$ represent the negative consequence of uncertainty. Figure 3 provides the intuition behind the three comparisons, where a smaller value of z_m 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 Eq. 1 as follows:

$$\tilde{S}(arch_1) \leq \tilde{S}(arch_2) \Leftrightarrow z_{m1} \leq z_{m2}, z_{p1} \geq z_{p2}, z_{n1} \leq z_{n2} \quad \text{Eq. 2}$$

The three comparisons on the right are formal representations of the three criteria from Section 3.3.

Our fuzzy comparison operator is an instance of a multi-dimensional comparison, 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 Figure 2b. Intuitively, 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 describe the details of these three steps.

5.1 Normalizing the Values Being Compared

Since the three z values are defined differently in terms of $\tilde{S}(arch)$, their range may not be the same. Therefore, to avoid one comparison to dominate the other ones as we combine them, we first have to normalize the z values. We use normalizing linear membership function [13], which is a function μ that maps each z to a value between 0 and 1:

$$\forall i \in \{m, p, n\}: \left(\mu_{z_i}: dom(z_i) \longrightarrow [0,1]\right)$$

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:

$$z_m^{PIS} \stackrel{\text{def}}{=} argmin_{(arch \in AS)}\, z_m$$
$$z_m^{NIS} \stackrel{\text{def}}{=} argmax_{(arch \in AS)}\, z_m$$
$$z_p^{PIS} \stackrel{\text{def}}{=} argmax_{(arch \in AS)}\, z_p$$
$$z_p^{NIS} \stackrel{\text{def}}{=} argmin_{(arch \in AS)}\, z_p$$
$$z_n^{PIS} \stackrel{\text{def}}{=} argmin_{(arch \in AS)}\, z_n$$
$$z_n^{NIS} \stackrel{\text{def}}{=} argmax_{(arch \in AS)}\, z_n$$

Note that the NIS and PIS definitions for z_m and z_n are reverse of that of z_p due to their semantic differences (i.e., we prefer a solution with small z_m 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:

$$\mu_{z_m} \begin{cases} 0 & z_m < z_m^{PIS} \\ \dfrac{z_m - z_m^{PIS}}{z_m^{NIS} - z_m^{PIS}} & z_m^{PIS} \leq z_m \leq z_m^{NIS} \\ 1 & z_m > z_m^{NIS} \end{cases}$$

$$\mu_{z_p} \begin{cases} 0 & z_p > z_p^{PIS} \\ \dfrac{z_p^{PIS} - z_p}{z_p^{PIS} - z_p^{NIS}} & z_p^{NIS} \leq z_p \leq z_p^{PIS} \\ 1 & z_p < z_p^{NIS} \end{cases}$$

Function μ_{z_n} is specified similar to μ_{z_m}. As the definitions of NIS and PIS are reversed, the normalizing function μ_{z_m} and μ_{z_n} are increasing, while $\mu_{z_p}$is decreasing.

Defining the normalization functions this way also allows us to flip the comparison for z_p. In other words, $\mu_{z_{p1}} \leq \mu_{z_{p2}}$ becomes the normalized equivalent of $z_{p1} \geq z_{p2}$. Thus, we rewrite Eq. 2 using the normalized values as follows:

$$\tilde{S}(arch_1) \leq \tilde{S}(arch_2) \Leftrightarrow \mu_{z_{m1}} \leq \mu_{z_{m2}}, \mu_{z_{p1}} \leq \mu_{z_{p2}}, \mu_{z_{n1}} \leq \mu_{z_{n2}} \quad \text{Eq. 3}$$

5.2 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 to transform the multi-dimensional comparison of Eq. 3 to a single-dimensional form [10]. We define φ to be the sum of the three normalized values as follows:

$$\varphi = \sum_{i \in \{m,p,n\}} \mu_{z_i} \quad \text{Eq. 4}$$

Therefore, $\tilde{S}(arch_1) \leq \tilde{S}(arch_2) \Rightarrow \varphi_1 \leq \varphi_2$. While this does not alone result in Eq. 3 to hold, $\varphi_1 \leq \varphi_2$ is a collective approximation of the extent in which $\tilde{S}(arch_1) \leq \tilde{S}(arch_2)$ [10][15]. In other words, for all practical purposes, if the value of φ_1 in $arch_1 \in AS$ is less than the value of φ_2 in $arch_2 \in$

AS, we say that $arch_1$ has a lower range, and therefore, better.

5.3 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 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_m, w_p, w_n, where $w_m+w_p+w_n=1$, to normalized values μ_{z_m}, μ_{z_p}, and μ_{z_n}, respectively. The weights specify the importance of each comparison relative to others. Thus, we rewrite Eq. 4 by including the weights as follows:

$$\varphi = \sum_{i\in\{m,p,n\}}(w_i\ \mu_{z_i}) \qquad \text{Eq. 5}$$

6. Exploring Solution Space under Uncertainty

The ability to compare architectures under uncertainty provides the foundation for architectural exploration and decision making in early design. In this section, we describe four ways in which GuideArch helps with this process.

6.1 Identifying Valid Architectures and Critical Constraints

GuideArch allows the architect 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 4.4. 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 to be less than $10,000. Determining whether this constraint is satisfied for a candidate architecture is challenged by the fact that the *Cost* of realizing that 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 from Section 4.4 as follows:

$$\forall p \in P_{max}: Thd_p \leq S_p^{<}(arch)$$

$$\forall p \in P_{min}: S_p^{<}(arch) \leq Thd_p$$

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 is not only going to ensure the architect does not pick a solution that is invalid, but can also be used to provide useful statistical measures. For instance, GuideArch provides the ratio of architectures comprising the solution space that are disqualified by each constraint. This allows the architect to identify both the limiting and loose constraints. Such information is crucial, as it allows the architect to explore the trade-offs in tightening or relaxing some of those constraints. If certain trade-offs are deemed appropriate, they could be communicated to other stakeholders, which could result in revising the constraints through appropriate negotiations.

6.2 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 4.5. We define this as a linear programming problem:

$$argmin_{(arch\in AS)}\ \varphi_{arch}$$

Subject to:

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

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

$$\forall p \in P_{max}: Thd_p \leq S_p^{<}(arch)$$

$$\forall p \in P_{min}: S_p^{>}(arch) \leq Thd_p$$

Here, the solver uses the approach described in Section 5 to compare the total value of properties between the candidate solutions. The architecture with the minimum value of φ (recall Figure 3 and Eq. 5) is the one that achieves the best combination of small anticipated value, small negative consequence of uncertainty, and large positive consequence of uncertainty.

6.3 Ranking the Architectures

The ability to find the optimal solution is complemented with the ability to see a ranking of candidates in GuideArch. 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 not to purchase 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. In other words, 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 5, to also rank them from best (top) to worst (bottom). We let R represent the ranking of top t architectures as an ordered list:

$$R \stackrel{\text{def}}{=} \langle arch_1, .., arch_t \rangle \text{ where } arch_i \in AS \wedge \left(\forall arch_i \in R: \nexists c \in AS \wedge c \notin R: \tilde{S}(c) \leq \tilde{S}(arch_i)\right) \wedge \left(\forall arch_i, arch_k \in R, i \leq k: \tilde{S}(arch_i) \leq \tilde{S}(arch_k)\right)$$

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

6.4 Identifying and Ranking the Critical Decisions

Modern software processes adopt an iterative process, where in each iteration the decisions made in the previous cycles are assessed and risks are mitigated [19]. 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 [19].

GuideArch could help the architect identify decisions that are likely to be most critical to the success of a software engineering project, and thus pose the greatest risk. The underlying insight is that a decision is likely to be crucial if it satisfies the following two criteria: (1) the alternatives selected from that decision have a large impact on the properties of architectures ranked at the top; and (2) the impact of those alternatives is highly uncertain. This is reasonable, since the architect is likely to select one of the top ranked architectures. Moreover, decisions with alternatives that have both large and uncertain impact on properties of the top ranked architectures introduce more error than others.

We quantify these two criteria as follows:

(1) *Impact on system properties*: 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:

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

The above approach gives equal weight to the choices made in the ranked architectures. However, since 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:

$$\tilde{E}_d = \sum_{r=1:t} \left(\tilde{E}_a \, e^{-\gamma r}\right), \text{ where } \gamma \text{ is the decay factor}$$

The larger γ, the more emphasis is placed on the alternatives appearing at the top of the ranking.

(2) *Uncertainty in the impact:* Unlike the ranking of architectural candidates, where we were 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). We let L represent the ranking of n most critical decisions as an ordered list:

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

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 reduce the uncertainty by investing resources and time (e.g., prototyping) to develop a better understanding of the alternatives' impact.

7. Evaluation

GuideArch was used by a team of engineers and academics to explore the architectural space of the EDS project (recall Section 2). In this section, we describe some of the salient outcomes of this study, as well as the results obtained through additional experiments in the laboratory.

7.1 Critical Constraints

One of the early requirements posed by client in EDS was to keep the cost of a single handheld device (which 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 6.1, GuideArch was able to show 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

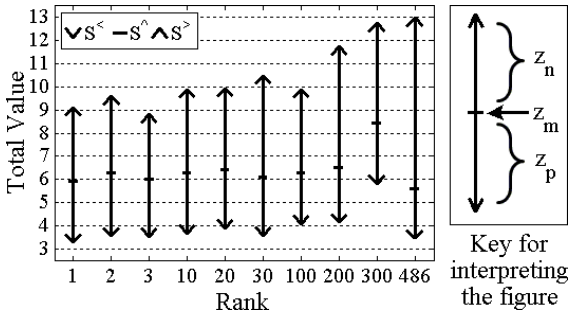

Figure 4. The triangular fuzzy value of 10 architectural candidates in EDS.

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 project and experiments reported here, the relaxed cost constraint was used.

7.2 Ranking

Figure 4 shows the triangular fuzzy value of 10 sample EDS architectures calculated by GuideArch. The horizontal axis marks the architectures' rankings. TABLE II shows the selected alternatives for the architectures in Figure 4. As you may recall from Section 5, one architecture is better than another if it has a lower z_m and z_n, and a larger z_p. In EDS, we gave equal weights to the satisfaction of each of those conditions (i.e., w_m=w_n=w_p=1/3). Looking at Figure 4 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_m and 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 in total.

7.3 Optimal Architecture

Upon further analysis, we observed that the traditional approaches would have selected the 486th candidate in Figure 4 as the optimal solution. Recall from Section 2 that traditional approaches do not consider uncertainty and only minimize the anticipated value (i.e., z_m). Comparing the difference between 1st and 486th candidates sheds light on the contributions of GuideArch. 486th architecture achieves the lowest z_m 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_m but with a better range of uncertainty.

The difference between 1st and 486th architecture could also be gleaned from the specific alternatives selected by each approach. For instance, looking up the two architectures in Tables II to find the selected alternatives, and subsequently looking up the alternatives in Table I, we can see that for the *Chat Protocol* decision, GuideArch selects *Openfire,* while traditional selects *In house*. *Openfire* is a COTS implementation of the XMPP protocol, while *In house* refers to the internal implementation of it. The EDS team was less certain with the *Response Time* property of *In house,* which had never been tested in large deployments with many clients. Therefore, as shown in Table I, *In house* had significantly larger pessimistic value for its *Response Time* (200ms for *In house* versus 70ms for *Openfire*). On the other hand, since *Openfire* was not optimized for smartphones, it had a slightly larger anticipated value for the *Response Time* (40ms for *In house* versus 60ms for *Openfire*). As a result, GuideArch selects *Openfire*, since it has a better range, i.e., makes a small compromise on anticipated value to prevent the large negative consequence of uncertainty. On the other hand, the traditional approach selects the solution with the best anticipated value, and ignores the high possibility of very bad *Response Time*.

7.4 Critical Decisions

GuideArch was also used to identify the critical decisions. Figure 5a shows the estimated impact of

TABLE II. THE ALTERNATIVES COMPRISING ARCHITECTURES SHOWN IN FIGURE 4.

Dimensions / Rank	Location Finding	Hardware Platform	File Sharing Package	Report Synchronization	Chat Protocol	Map Access	Connectivity	Database	Architectural Style	Data Exchange format
1	2	1	2	1	1	3	1	1	3	3
2	2	1	2	1	1	3	1	1	3	1
3	2	1	1	1	1	3	1	1	3	3
10	2	2	2	1	1	3	1	1	3	1
20	2	2	2	1	1	3	1	1	3	2
30	2	1	1	1	2	3	1	1	3	1
100	2	1	1	1	1	3	1	2	3	1
200	2	1	2	2	1	3	4	1	2	2
300	2	1	2	1	1	1	1	2	1	2
486	2	2	1	1	2	3	4	1	2	3

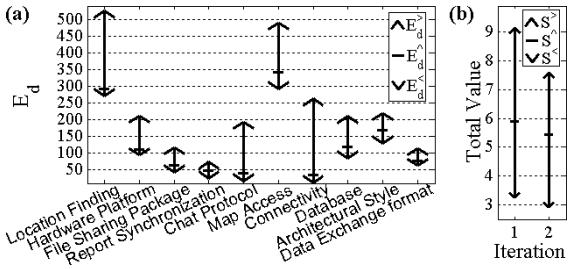

Figure 5. Critical decisions: (a) comparing the decisions in the first iteration and (b) improvement in the optimal architecture after revising the critical decision in the subsequent iteration.

decisions (i.e., $\tilde{E}_d$) as well as the range of uncertainty in those estimates, which as you may recall from Section 6.4 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. In other words, if we use the fuzzy comparison operator (recall Section 5), *Location Finding*'s $\tilde{E}_d$ is larger among all other decisions. The reason behind this can be gleaned from Table I: both of *Location Finding*'s alternatives (*GPS* and *Radio Triangulation*) have a large and uncertain impact on two of the properties with the highest priority (*Battery Usage* and *Response Time*).

This analysis formed the first iteration of using GuideArch. It helped the team identify the critical decisions early on, and focus additional efforts on them. As a result, the team came across an advanced state-of-the-art variation of the traditional radio triangulation [14] that presented the project with a new alternative for *Location Finding*. Given that a prototype of this solution had been developed, evaluated, and published [14], 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. Figure 5b shows the range of total value (i.e., the fuzzy value of $\tilde{S}$) for the best candidate architecture picked by GuideArch in this second

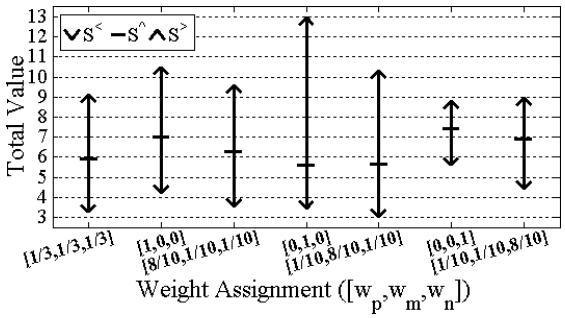

Figure 6. The optimal architecture for different weight assignments.

iteration, compared to the best candidate picked in the first iteration. As Figure 5b shows, the introduction of the new alternative (i.e., smart radio triangulation) improved the total value of the optimal architecture significantly.

7.5 Tuning GuideArch

In the EDS project we gave the same weight to the three comparisons (i.e., w_m=w_n=w_p=1/3). However, recall from Section 5.3 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 EDS model to assess the impact of weights on the optimal architecture selected by GuideArch. Figure 6 shows the results executed in the model used in the first iteration and before the introduction of additional alternatives discussed in the previous section.

To allow for comparison, in Figure 6, 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 Figure 4. Also note that when w_m=1 and w_n=w_p=0, GuideArch behaves exactly like the traditional approach. Therefore, the optimal architecture for that weight assignment in Figure 6 (i.e., [0,1,0]) is the same as candidate ranked 486th in Figure 4. 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. It is important to note that no particular weight assignment in Figure 6 is the best. We can envision situations in which placing emphasis on one of the comparisons may be more appropriate, which GuideArch allows for naturally.

7.6 Performance Benchmark

Finding the optimal architecture in the EDS project took 281ms. Fast response allowed the EDS team to rapidly perform numerous "what if" scenario analyses by making changes to the model. For a better illustration of GuideArch's performance, we augmented our experience in EDS with benchmarks depicted in Figure 7. Here we compare the execution time of the traditional approach with that of GuideArch on a PC with Intel® Core™ 2 Duo CPU and 3.50 GB of RAM running Microsoft® Windows® XP SP 3. It took GuideArch longer to find the optimal

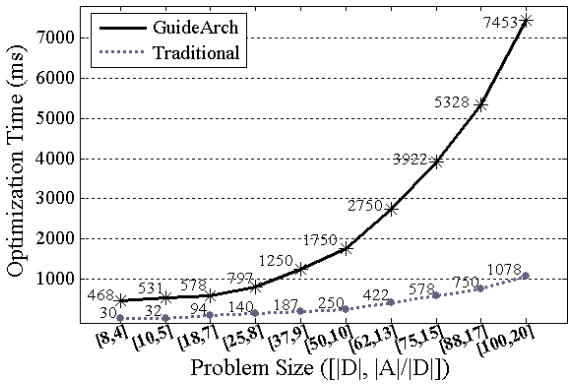

Figure 7. Performance of GuideArch compared to traditional approaches. |D| is the number of decisions and |A|/|D| is the average number of alternatives per decision in the problem.

solution, which is not surprising, since as you may recall from Section 5.1, GuideArch requires 6 additional optimizations to calculate PIS and NIS value pairs for the three z values. Figure 7 corroborates this as the performance of GuideArch is higher than the performance of traditional approach by a constant factor of 7. While it takes longer to execute GuideArch, we believe its performance to be reasonably fast for most projects. It took 7.5 seconds to find the optimal configuration in the largest problem, consisting of 100 decisions and average of 20 alternatives per decision, for a total of $20^{100} = 1.2 \times 10^{130}$ possible combinations.

8. Related Work

Making architectural decisions is a problem that has been studied from both design-time and run-time perspectives. Here we discuss only those targeted at design-time, since that has been the focus of our work:

- ArchDesigner [2] is a quality-driven approach to find an optimal architecture that meets conflicting stakeholders' quality goals. ArchDesigner uses linear programming to find the optimal architecture.
- CBAM [12] is a quantitative approach for economic modeling of software engineering decisions, which builds upon ATAM [6]. CBAM provides the cost and benefit of different architectural candidates.
- Cortellessa et al. dealt with the problem of COTS selection in [7], which helps the architects decide if a component should be purchased or developed in house. They formulate this as an optimization problem.
- ArcheOpterix [1] is a tool for optimizing an embedded system's architecture. It uses evolutionary algorithms for multi-objective optimization of such systems.
- In [15], we dealt with the problem of selecting a deployment architecture (i.e., mapping of components to hardware nodes), such that multiple QoS objectives are maximized.

While many of the above approaches [12][2][15] acknowledge the challenges posed by uncertainty, none addresses it explicitly and in the form of a mathematical framework for decision making.

A few approaches have considered uncertainty:

- Andreou and Papatheocharous [3] pass data from past projects through Fuzzy Decision Trees to build association rules that approximate the cost of software.
- Palladio [4] uses information about components comprising the architecture to derive analytical models (e.g., Queueing Networks) and simulate the system's performance. Random variables are used to specify uncertainty in service demands and iterations.

[3][4] are complementary to GuideArch, as they could be used to specify the range of cost and performance, respectively. Unlike GuideArch, [3][4] neither optimize a range of values, nor have the ability to compare ranges of possible behaviors for different architectures.

Finally, in [8], Doyle et al. present an approach to compare alternative architectures, assuming the availability of a probability distribution representing the response time of each architecture. Instead of fuzzy mathematics, they rely on extensive integration to compare the probability distributions, which are computationally expensive, making their approach inapplicable to exploration of a large solution space. Moreover, their approach is specific to response time.

9. Conclusion and Future Work

In any software project, early architectural decisions represent some of the most important decisions engineers ever make. Yet there is a lack of techniques and tools for helping the engineers 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 of 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 a case study as well as the laboratory experiments helped us to experience its benefits firsthand, it also suggested

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. In other words, we currently assume all stakeholders have agreed on the estimated impact of alternatives on properties and their priorities. However, this may not always be the case, presenting GuideArch with yet another source of uncertainty. Another avenue of future work is to extend GuideArch to the kinds of uncertainty discussed in Section 4.6.

10. Acknowledgments

This work is partially supported by grant CCF-0820060 from the National Science Foundation. We want to thank Mr. Ehsan Kouroshfar and Mr. Thabet Kaczem for their helps.

11. References

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

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

[3] Andreou, A.S. and Papatheocharous, E. 2008. Software Cost Estimation using Fuzzy Decision Trees. *Int'l Conf on Automated Software Engineering* (L'Aquila, Italy, Sep. 2008), 371–374.

[4] Becker, S., Koziolek, H. and Reussner, R. 2009. The Palladio component model for model-driven performance prediction. *J. Syst. Softw.* 82, 1 (Jan. 2009), 3-22.

[5] Brooks, F.P. 1995. *The Mythical Man-Month: Essays on Software Engineering, Second Edition*. Addison-Wesley Professional.

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

[7] Cortellessa, V., Marinelli, F. and Potena, P. 2008. An optimization framework for "build-or-buy" decisions in software architecture. *Computers & Operations Research*. 35, 10 (Oct. 2008), 3090-3106.

[8] Doyle, G.S. 2010. *A Methodology for Making Early Comparative Architecture Performance Evaluations*. PhD Dissertation, George Mason University.

[9] Dubois, D., Prade, H. and Sandri, S. 1993. On possibility/probability transformations. *IFSA Conference* (Seoul, Korea, Jul. 1993).

[10] Facchinetti, G. and Ghiselli Ricci, R. 2004. A characterization of a general class of ranking functions on triangular fuzzy numbers. *Fuzzy Sets and Systems*. 146, 2 (Sep. 2004), 297-312.

[11] Garlan, D. 2010. Software Engineering in an Uncertain World. *FSE/SDP Wrkshp. on the Future of Software Engineering Research* (Santa Fe, New Mexico, Nov. 2010).

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

[13] Lai, Y.-J. and Hwang, C.-L. 1992. A new approach to some possibilistic linear programming problems. *Fuzzy Sets Syst.* 49, 2 (Jul. 1992), 121-133.

[14] Lin, K., Kansal, A., Lymberopoulos, D. and Zhao, F. 2010. Energy-accuracy trade-off for continuous mobile device location. *Proceedings of the 8th international conference on Mobile systems, applications, and services* (San Francisco, California, Jun. 2010), 285–298.

[15] Malek, S., Medvidovic, N. and Mikic-Rakic, M. An Extensible Framework for Improving a Distributed Software System's Deployment Architecture. *accepted to appear in IEEE Trans. Softw. Eng.*

[16] Malek, S., Mikic-Rakic, M. and Medvidovic, N. 2005. A Style-Aware Architectural Middleware for Resource-Constrained, Distributed Systems. *IEEE Trans. Softw. Eng.* 31, 3 (Mar. 2005), 256-272.

[17] Meier, R. 2008. *Professional Android application development*. Wiley-India.

[18] Open Handset Alliance: *http://www.openhandsetalliance.com/*.

[19] Yang, Y. and Boehm, B. 2007. Improving process decisions in COTS-based development via risk-based prioritization. *Software Process: Improvement and Practice*. 12, 5 (Sep. 2007), 449-460.

[20] Zadeh, L.A. 1999. Fuzzy sets as a basis for a theory of possibility. *Fuzzy Sets Syst.* 100, (Jun. 1999), 9-34.

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