Uncertainty, Risk, and Information Value in Software Requirements and Architecture

Emmanuel Letier David Stefan Earl T. Barr
Department of Computer Science
University College London
London, United Kingdom
{e.letier, d.stefan, e.barr}@ucl.ac.uk

ABSTRACT

Uncertainty complicates early requirements and architecture decisions and may expose a software project to significant risk. Yet software architects lack support for evaluating uncertainty, its impact on risk, and the value of reducing uncertainty before making critical decisions. We propose to apply decision analysis and multi-objective optimisation techniques to provide such support. We present a systematic method allowing software architects to describe uncertainty about the impact of alternatives on stakeholders' goals; to calculate the consequences of uncertainty through Monte-Carlo simulation; to shortlist candidate architectures based on expected costs, benefits and risks; and to assess the value of obtaining additional information before deciding. We demonstrate our method on the design of a system for coordinating emergency response teams. Our approach highlights the need for requirements engineering and software cost estimation methods to disclose uncertainty instead of hiding it.

Categories and Subject Descriptors

D2.11 [**Software Engineering**]: Software Architectures

General Terms

Design, Economics, Theory

Keywords

Software engineering decision analysis

1. INTRODUCTION

Uncertainty is inevitable in software engineering. It is particularly present in the early stages of software development when an organisation needs to make strategic decisions about which IT projects to fund, or when software architects need to make decisions about the overall organisation of a software system. In general, these decisions aim at maximising the benefits that the software system will bring to its stakeholders, subject to cost and time constraints. Uncertainty includes uncertainty about stakeholders' goals and their priorities, about the impact of alternatives on these goals, about the feasibility, cost, and duration of implementing the alternatives, about future changes in stakeholders' goals, business context and technological environments, and finally uncertainty about whether the right questions about decisions are even being asked and all their options identified.

In a decision problem, uncertainty is a lack of complete knowledge about the actual consequences of alternatives. For example, software architects may be uncertain about the cost and performance impact of a proposed software architecture. Given their current knowledge, they might estimate the cost to be between £1m to £3m and the achievable response time to be between 1 and 10 seconds. A risk exists when the possible consequences of a decision include undesirable outcomes, like loss or disaster [40]. Continuing the example, selecting the proposed architecture might carry the risks of the development costs exceeding £2m and the response time not achieving the minimally acceptable target of 2 seconds. In software architecture decisions, the risks include selecting an architecture that is too expensive to develop, operate, and maintain, that is delivered too late and, most importantly, that fails to deliver the expected benefits to its stakeholders. Numerous studies have shown that these risks are severely underestimated [29]. This is not surprising: uncertainty and risks are rarely considered explicitly in software engineering decisions and the software engineering literature offers no principled approaches to deal with them.

In this paper, we focus on early requirements and architecture decisions, i.e. decisions about the functionality the software should provide, the quality requirements it should satisfy, its organisation into components and connectors, and its deployment topology. We assume stakeholders' goals and the alternatives have been identified using appropriate requirements engineering and software architecture methods [45, 47, 63, 65]. Our objective is to support reasoning about uncertainty concerning the impact of alternatives on stakeholders' goals.

Previous work dealing with uncertainty in early requirements and architecture decisions [21, 42, 49, 62] suffers important limitations: they use unreliable methods for eliciting uncertainties (some confuse group consensus with certainty); they tend to evaluate alternatives against vague, unfalsifiable criteria; they provide no information about the risks that accompany uncertainty; and they provide no support for assessing to what extent obtaining additional information before making a decision could reduce these risks.

We address these limitations by adapting concepts and techniques from statistical decision analysis to the problems of early requirements and architecture design decisions. Decision analysis is a discipline aiming at supporting complex decisions under uncertainty with systematic methods and mathematical tools for understanding, formalising, analysing, and providing insights about the decision problem [38]. Decision analysis is used notably in the health care

ICSE '14, May 31 – June 7, 2014, Hyderabad, India

domain to inform decisions about the cost-effectiveness of new medical treatments based on the results of clinical trials [7]. There are exceptional uses of these methods in the context of IT investment decisions [14, 41], but despite their relevance to early requirements engineering and architecture decisions, they have been largely ignored by the software engineering community.

Our approach to early requirements and architecture decisions consists in formalising the decision problem in terms domain-specific measurable goals, to elicit and represent uncertainties as probability distributions, to simulate the impact of alternatives on goals through Monte-Carlo (MC) simulations, and to shortlist a set of alternatives using Pareto-based multi-objective optimisation techniques. We introduce the software engineering community to the *value of information*, a powerful notion from decision analysis, that allows a decision maker faced with uncertainty to measure those uncertainties and determine which would be most profitably reduced.

The paper's main contribution is a systematic method for applying statistical decision analysis techniques to early requirements and architecture decision problems (Section 3). By developing this method, we were also led to make the following contributions:

1. We define novel decision risk metrics tailored for requirements and architecture decision problems (Section 3.3);
2. We extend the concept of value of information, traditionally defined in terms of impact of additional information on expected outcomes only, by considering how additional information reduces risk (Section 2.3).
3. We introduce the concept of Pareto-optimal *strip*, a generalisation of a Pareto-optimal front, designed to resist modelling and measurement errors present in multi-objective decisoin problem under uncertainty (Section 3.5);

We have developed a tool supporting our approach and have applied it to data from a real system from the literature [21]. Our tool and all models discussed in this paper are available at `www.cs.ucl.ac.uk/staff/e.letier/sdda`.

2. COST-BENEFIT ANALYSIS UNDER UNCERTAINTY

Before considering early requirements and architecture decision problems, we first consider the simpler problem of selecting one alternative among a set of alternatives based on their costs and benefits. Such problem assume a model exists to calculate the costs and benefits of all alternatives in a common unit, which is usually monetary (*e.g.* Pound, Euro, Dollar, Yen or Rupee) [7]:

Definition. A *cost-benefit decision model* comprises a set A of alternatives, a set Ω of model parameters, and two functions, $cost(a,\omega)$ and $benefit(a,\omega)$, that return the cost and benefit of alternative a given the parameter values ω. The net benefit of an alternative is then $NB(a,\omega) = benefit(a,\omega) - cost(a,\omega)$. To simplify the notation, we sometimes leave the model parameters implicit and write $NB(a)$ for $NB(a,\omega)$, and similarly $benefit(a)$ and $cost(a)$.

Example. An engineering firm is considering replacing an ageing Computer-Aided Design (CAD) application with a new system. The set of alternatives is $A = \{\text{legacy}, \text{new}\}$. The CAD application helps the firm design complicated engineering artefacts (*e.g.* turbines, aircraft engines, *etc.*) that it sells to clients. The benefits associated with each alternative $a \in A$ is a function of several variables such as the market size, the market share that each alternative might help achieving, which itself is a function of features of each CAD. Likewise, the cost associated with each alternative is a function of several parameters such as the development, maintenance and operational costs. The cost and benefit functions would typically also include concerns related to incremental benefit delivery, cash flow, and discount factors [14, 16]. The model parameters are the variables in these equations, *i.e.* those that are not further defined in terms of other variables. To keep our illustrative example simple, we will hide the details of the cost and benefit functions and discuss decisions based on the results of these functions only.

2.1 Computing Expected Net Benefit and Risk

Traditional Cost-Benefit Anlysis (CBA) computes the net benefit of each alternative using point estimates (exact numbers instead of ranges) for each of a model's parameter. Such approaches therefore ignore the often large uncertainty about parameter values. Uncertainty about cost and benefit exists but is hidden. In a statistical CBA, uncertainty about the model parameters is modelled explicitly as probability distributions and used to compute the probability distribution of net benefit for each alternative.

Simple, effective methods exist for eliciting the model parameters' probability distributions from decision makers [52]. These methods have sound mathematical foundations and are based on significant empirical studies of how uncertainty can be reliably elicited from humans. We will not be concerned with these methods in this paper beyond noting that they can and should be used to elicit reliable probability distributions from domain experts and decision makers.

Once the model parameters probability distributions have been estimated, one needs to compute the probability distributions for the cost, benefit and net benefit of each alternative. It is generally not possible to compute these probability distributions analytically because the model equations and parameters' probability distributions can be arbitrarily complicated. Monte-Carlo (MC) simulations can, however, compute good approximations. The underlying principle is to sample a large number of simulation scenarios generated by model parameter values drawn from their probability distributions and use them to compute the net benefit in that scenario. The result of a MC simulation of a cost-benefit decision model is a $M \times N$ matrix $\widehat{NB}$ where M is the number of simulated scenarios and N is the number of alternatives in A. The element $\widehat{NB}[i,j]$ denotes the net benefit for alternative j in the i^{th} scenario.

From the result of a MC simulation, one can, for each alternative, estimate measures of interest to decision makers such as the expected net benefit (*ENB*), loss probability (*LP*), and probable loss magnitude (*PLM*), defined as follows:

$$\begin{aligned} ENB(a) &= E[NB(a)] \\ LP(a) &= P(NB(a) < 0) \\ PLM(a) &= E[NB(a) | NB(a) < 0] \end{aligned}$$

where $E[X]$ denotes the expectation of a random variable X.

Example. Figure 1 shows the results of a statistical CBA for our illustrative example. We assume cost and benefit have a normal distribution truncated at zero. Figure 1a shows the mean and 90% confidence interval of these distributions. A 90% confidence interval means that decision makers believe there is a 90% chance that the actual costs and benefits will fall within these ranges. Figure 1b shows the resulting expected net benefit, loss probability, and probable loss magnitude of each alternative. It shows developing the new CAD has a high expected net benefit but also high risks in terms of loss probability and probable loss magnitude. In a traditional CBA, these risks would not have been quantified and would, most likely, have been underestimated if not entirely ignored.

2.2 The Expected Value of Information

If, before making a decision, decision makers could pay someone to obtain additional information that reduce uncertainty about the cost and benefits of alternatives, how much would that information

	mean	90% CI
benefit(new)	£5m	[£1m–£9m]
cost(new)	£3m	[£1m–£5m]
benefit(legacy)	£1m	[£0.9m–£1.1m]
cost(legacy)	0	[0m–0m]

(a) Mean and 90% Confidence Intervals (CI).

	new	legacy
ENB	£2m	£1m
LP	23%	0%
PLM	£1.4m	0

(b) Expected Net Benefit (*ENB*), Loss Probability (*LP*), and Probable Loss Magnitude (*PLM*).

	EVPPI	$\Delta_{PPI}(LP)$
benefit(new)	£0.54m	8%
cost(new)	£0.14m	4%
benefit(legacy)	£0	0%
cost(legacy)	£0	0%

(c) Information Value Analysis. The value of total perfect information (*EVTPI*) is £0.64m.

Figure 1: Statistical Cost-Benefit Analysis for deciding whether to replace a legacy application by a new system.

be worth to them? It is possible to answer this question by computing the expected value of information [37]. Intuitively, information that reduces uncertainty may lead decision makers to select an other alternative with highest expected net benefit than the alternative they would select without additional information. The expected value of information is the expected gain in net benefit between the selected alternatives with and without the additional information.

The expected value of information for the different model variables tells decision makers to focus on reducing uncertainty about information with high expected value and to avoid wasting effort reducing uncertainty about information with low expected value (or at least not pay more for information than its expected value). Computing the expected value of information can yield surprising results. Hubbard reports he has applied information value theory to 20 IT project business cases (each having between 40 to 80 variables) and observed the following pattern: (1) the majority of variables had an information value of zero; (2) the variables that had high information value were routinely those that the client never measured; (3) the variables that clients used to spend the most time measuring were usually those with a very low (even zero) information value [41]. The contrast between the second and third observations constitutes what Hubbard has called the IT measurement inversion paradox [39]. He cites the large effort spent by one of his clients on function point analysis [4] — a popular software development productivity and cost estimation method — as an example of measurement with very low information value because its cost estimation were not more accurate or precise than the project managers' initial estimates.

The expected value of information is defined with respect to an outcome to be maximised and assumes a default decision strategy of maximising expected outcome. In this section, the outcome to be maximised is the net benefit *NB* but the definition applies to any other outcome, *e.g.* maximising software reliability. Information is valued in the same units as the outcome that it measures. This makes measuring information value with respect to net benefit particularly attractive as it assigns a financial value to information.

The expected value of *total* perfect information (*EVTPI*) is the expected gain in net benefit from using perfect information about all model parameters:

$$EVTPI = E[\max_{a \in A} NB(a, \omega)] - \max_{a \in A} E[NB(a, \omega)].$$

In this definition, the second term denotes the highest expected net benefit given the current uncertainty about the model parameters Ω, and the first term the expectation over all possible values ω for the parameters Ω of the highest net benefit when the parameter values are ω (in other words, the expected net benefit from obtaining perfect information). Observe how the two terms invert the application of the expectation and maximisation operators. It can be shown that EVTPI is always positive or zero. The *EVTPI* can be estimated from the output $\widehat{NB}$ of a MC simulation:

$$EVTPI = \underset{i:1..N}{\text{mean}} \max_{j:1..M} \widehat{NB}[i, j] - \max_{j:1..M} \underset{i:1..N}{\text{mean}} \widehat{NB}[i, j].$$

As an illustration, Figure 2 shows how EVTPI is computed from a small MC simulation with 5 scenarios (the actual MC simulation used to produce the results in Figure 1 consists of 10^4 scenarios).

Information value theory also defines the expected value of *partial* perfect information, *i.e.* perfect information about a subset of the model parameters [25, 58], and the expected value of (partial or total) *imperfect* information, *i.e.* information that reduces uncertainty but without completely eliminating it [41]. In this paper, we only use the expected value of perfect information either in *total*, over all parameters, or about a *single* model parameter. The expected value of imperfect information and of perfect information about sets of parameters are harder to compute, and they may not yield substantial practical benefits over simpler information value analysis.

The expected value of *partial* perfect information about a *single* model parameter Θ, noted $EVPPI(\Theta)$, is the expected gain in net benefit from using perfect information about Θ:

$$EVPPI(\Theta) = E[\max_{a \in A} f(a, \theta)] - \max_{a \in A} E[NB(a, \omega)],$$

where $f(a, \theta) = E_{\Omega - \Theta} NB(a, \omega)$ is the expected *NB* of alternative a, conditioned on the parameter Θ fixed at θ, and $E_{\Omega - \Theta}$ denotes the expectation with respect to all model parameters in Ω except Θ [58]. The intuition of this definition is similar to that of EVTPI. As EVTPI, it can be shown that *EVPPI* is always positive or zero.

Computing $EVPPI(\Theta)$ is harder than computing *EVTPI*. In this paper, we rely on a recent efficient algorithm that computes *EVPPI* by taking as input only the pair $\langle \widehat{\Theta}, \widehat{NB} \rangle$ of simulations for the model parameter Θ and the corresponding matrix of *NB* simulations generated by the MC simulation [58]. This algorithm first finds a suitable segmentation of values in $\widehat{\Theta}$ such that, within each segment, the differences in maximal expected *NB* remain small. For each segment, it computes the average gain in *NB* from knowing θ, then averages these average gains weighted by the proportion of simulation that falls into each segment.

Note that *EVTPI* and *EVPPI* compute the expected value of information about some parameters *before the value for these parameters are revealed*. Once the actual values are revealed, they may increase *or decrease* expected net benefit. The *EVTPI* and *EVPPI* merely compute how much the expected net benefit will change *on average*. It is these averages that are always positive or zero. The revelation of new information can also both increase or decrease uncertainty about the parameters' true values. When this happens, an increase of uncertainty is most likely caused by a failure to mitigate overconfidence biases during the elicitation of probability distributions. For example, decision makers with overconfidence biases will express 90% confidence intervals that are narrower than their true uncertainty. Overconfidence bias is a serious problem because the computations of expected net benefit, risk, and information value all assume the initial probability distributions are accurate. This observation reinforces the importance of using appropriate uncertainty elicitation techniques designed to reduce the effects of overconfidence and other biases [52].

Measuring expected information value is an alternative to sensitivity analysis. There are several variants of sensitivity analysis.

Scenarios	$\widehat{NB}$(new)	$\widehat{NB}$(legacy)	Max
1	£1.33m	£1.03m	£1.33m
2	£0.13m	£0.96m	£0.96m
3	£4.05m	£1.00m	£4.05m
4	£6.13m	£1.06m	£6.13m
5	-£1.39m	£1.07m	£1.07m
Mean	£2.05m	£1.02m	£2.71m

Figure 2: Illustration of a MC simulation and computation of EVTPI. The second and third columns show the $\widehat{NB}$ for the new and legacy applications in 5 random scenarios. Over these five scenarios, the new application has the highest expected net benefit (£2.05m). The fourth column show the maximal possible net benefit in each scenario and its mean value over all five scenarios (£2.71m). We thus have $EVTPI$ = £2.71m - £2.05m = £0.66m.

Possibly the most common in software engineering consists in measuring, for each model parameter taken individually, the change of *NB* (or other objective of interest) for a selected alternative when the parameter varies between some low and high value [49]. A parameter is then said to have high sensitivity if the changes in *NB* are high. The expected value of information differs from sensitivity analysis in that it takes into account the probability of changes in parameters' values and possible changes of alternatives to optimise net benefit. These differences have important implications: a parameter with high sensitivity may have low information value if it has a low probability of change; and vice-versa, a parameter with low sensitivity may have high information value if a highly probable change for this parameter leads to selecting a different alternative with much higher *NB*. Felli and Hazen provide a more detailed analysis of the benefits of measuring expected value of perfect information over sensitivity analysis [25].

2.3 The Impact of Information on Risk

Using additional information to maximise expected *NB* impacts risk, sometimes favourably, as when selecting an alternative with highest expected *NB* also reduces risk, or unfavourably, as when selecting an alternative with highest *NB* increases risk. Measuring this impact gives decision makers additional information about the value of seeking additional information.

We have thus defined a new measure of the expected impact of perfect information on risk. In our cost-benefit analysis, risk is measured by the loss probability and the probable loss magnitude. To keep the exposition simple, we define the impact of perfect information on risk with respect the a risk measure $Risk(a) = P(F(a,\omega))$ where $F(a,\omega)$ is true when alternative a fails when parameter values are ω. For example, for $LP(a)$, $F(a,\omega)$ is $NB(a,\omega) < 0$. Our definition can easily be extended to risk measures, such as *PLM*, defined over real-valued rather than boolean F functions.

Let $a^\star$ be an alternative that maximises expected *NB*. If there is more than one alternative with equal highest expected *NB*, $a^\star$ is one with minimal risk. Let $a^\star(\omega)$ and $a^\star(\theta)$ be alternatives that maximise *NB* when $\Omega = \omega$ or $\Theta = \theta$, respectively. The expected impact of total (respectively, partial) perfect information on *Risk* is the expected difference between $Risk(a^\star(\omega))$ (respectively, $Risk(a^\star(\theta))$) and $Risk(a^\star)$:

$$\Delta_{TPI}(Risk) = E[Risk(a^\star(\omega))] - Risk(a^\star)$$
$$\Delta_{PPI(\Theta)}(Risk) = E[Risk(a^\star(\theta))] - Risk(a^\star).$$

The $\Delta_{TPI}(Risk)$ can be estimated from matrices $\widehat{NB}$ and $\widehat{F}$ generated during the Monte-Carlo simulation:

$$\Delta_{TPI}(Risk) = \underset{i:1..N}{\text{mean}}[\widehat{F}(which.max_{j:1..M}\widehat{NB}[i,j])] - \underset{i:1..N}{\text{mean}}[0 > \widehat{NB}[i,a^\star]].$$

where $which.max_{j:1..M}\widehat{NB}[i,j]$ denotes the column indices of the alternative with highest benefit in row i.

To compute $\Delta_{PPI(\Theta)}(Risk)$, we have extended the algorithm for computing *EVPPI* from the Monte-Carlo simulation data $\langle\widehat{\Theta},\widehat{NB},\widehat{F}\rangle$. Our extension applies the same principle as the one used to compute $\Delta_{TPI}(Risk)$) to compute the Δ in *Risk* in each segment of Θ values, then returns the weighted average of those Δ over all segments.

Example Figure 1-c shows the expected value of information in our illustrative example. The EVTPI is £0.64m, 32% of expected net benefit. Measuring EVPPI shows that reducing uncertainty about the new application's benefits has high value and reduces most of the risks, whereas reducing uncertainty about its cost has almost no value and little impact in reducing loss probability.

3. SOFTWARE DESIGN DECISIONS UNDER UNCERTAINTY

Software design decisions are usually more complex than the simple cost-benefit decision problems of the previous section. Complexity arises in the solution space, in the objective space, and in the models that relate the two.

In the solution space, instead of involving the selection of one alternative from a set, they typically involve a multitude of interrelated design decisions concerning choices among alternative architectural styles, design patterns, technologies, and responsibility assignments [63, 65]. This leads to an exponential increase in the number of candidate solutions; for example, if the problem involves 10 design decisions with 3 options each, the number of candidate architectures is 3^{10} (around 60,000). The solution space for software design decisions is therefore several orders of magnitudes larger than the solution spaces of other domains applying decision analysis techniques — for example, in healthcare economics the solution space rarely exceeds 5 different treatment options [7].

In the objective space, software design decisions typically involve multiple goals that are generally conflicting, hard to define precisely, and not easily comparable (unlike cost and benefit, they have different units of measure). Examples of goals include concerns related to security, performance, reliability, usability, and the improved business outcomes generated by the software. Clarifying these goals and understanding their trade-offs is a significant part of supporting software design decisions. The goals in healthcare decision problems are at least as complex as software design decision goals. There has, however, been a much greater effort at defining these goals and their trade-offs than for software engineering problems. This has resulted in measures such as the quality-adjusted life year used to compare alternative treatment options [7].

The models relating the design decision options to stakeholders' goals are often hard to build, validate, and include a very large number of parameters. They are typically composed of models of the software system (to evaluate the impact of software design decisions on software qualities such as its performance and reliability) and models of the application domain (to evaluate the impact of software and system design decisions on stakeholders goals).

To deal with this complexity, we propose the following process:

1. Defining the architecture decision model
2. Defining a cost-benefit decision model
3. Defining the decision risks

4. Eliciting parameters values
5. Shortlisting candidate architectures
6. Identifying closed and open design decisions
7. Computing expected information value

Steps 1 and 2 correspond to standard model elaboration activities performed notably in the ATAM [45] and CBAM [42, 49] approaches. Steps 3 and 4 are specific to architecture decisions under uncertainty. Step 5 extends Pareto-based muliobjective optimisation techniques to decisions under uncertainty. Step 6 identifies closed and open design decisions from this shortlist. Step 7 computes expected information values. At the end of these steps, if some model parameters or variables have high expected information value, software architects may choose to elicit further information and refine corresponding parts of their models to improve their decisions and reduce their risks. In practice, some of these steps may be intertwined. For example, the elaboration of the architecture decision model and the cost benefit model in steps 1 and 2 are likely to be interleaved rather than performed sequentially [51].

SAS Case Study. We apply our method on a case study of software architecture decisions presented at ICSE 2013 [21].

The software to be designed is a *Situational Awarness System* (SAS) whose purpose is to support the deployment of personnel in emergency response scenarios such as natural disasters or large scale riots. SAS applications would run on Android devices carried by emergency crews and would allow them to share and obtain an assessment of the situation in real-time (e.g., interactive overlay on maps), and to coordinate with one another (e.g., send reports, chat, and share video streams).

A team of academics and engineers from a government agency previously identified a set of design decisions, options and goals to be achieved by this system (see Figure 3). They also defined models for computing the impact of options on the goals and documented uncertainty about model parameters using three point estimates, a method commonly used by engineers and project managers that consists in eliciting a pessimistic, most likely, and optimistic value for each model parameter. They then applied a fuzzy-logic based approach, called GuideArch, to support design decisions under uncertainty [21].

To facilitate comparison between the approaches, we will apply our method on the same model and data as those used by the GuideArch method [20].

Decisions	Options
Location Finding	GPS Radio Triangulation
File Sharing	OpenIntents In house
Report Syncing	Explicit Implicit
Chat Protocol	XMPP (Open Fire) In house
Map Access	On demand (Google) Cached on server Preloaded (ESRI)
Hardware Platform	Nexus I (HTC) Droid (Motorola)
Connectivity	Wi-FI 3G on Nexus I 3G on Droid Bluetooth
Database	MySQL sqLite
Architectural Pattern	Facade Peer-to-peer Push-based
Data Exchange Format	XML Compressed XML Unformatted data

Goals
Battery Usage
Response Time
Reliability
Ramp Up Time
Cost
Development Time
Deployment Time

Figure 3: Overview of the SAS Case Study [21].

3.1 Defining the Architecture Decision Model

The first step consists in identifying the decisions to be taken together with their options, defining the goals against which to evaluate the decisions, and developing a decision model relating alternative options to the goals [35, 45]. The result is a multi-objective architecture decision model (MOADM).

Definition. A *multi-objective architecture decision model* is a tuple (D, C, Ω, G, v), where

- D is a set of design decisions where each decision $d \in D$ has several options O_d; a *candidate architecture* is a function $a : D \rightarrow \cup_{d \in D} O_d$ that maps each decision d to a single option in O_d; the set of all candidate architectures is noted A[1];
- C is a set predicates capturing dependency constraints between design decisions such as prerequisite, mutual exclusion, and mutual inclusion relations [59, 69]);
- Ω is a set of model parameters;
- G is a set of optimisation goals, partitioned into G+ and G- denoting goals to be maximised and minimized, respectively;
- v is a goal evaluation function such that $v(g, a, \omega)$ is a real value denoting the level of attainment of goal g by candidate architecture a when the model parameters have the concrete values ω.

Optimisation goals include software quality attributes such as performance, reliability, *etc.* and stakeholders goals such as the number of lives saved and property damage avoided during an emergency response. Software quality evaluation models (*e.g.* performance and reliability models) and quantitative goal-oriented requirements models [35, 47] are typical examples of goal evaluation functions. These models have parameters, such as the reliability of each component in a reliability block diagram or the likelihoods of different types of events requiring a coordinated emergency response in a quantitative goal model. In the standard use of these models, each parameter is assigned a point-based estimate. In step 4 of our method, the parameters are assigned probability distributions.

In goal-oriented requirements models [35, 47], candidate architectures describe socio-technical systems, *i.e.* systems for which components include human agents and hardware devices, as well as software components. The design decisions include decisions about alternative goal refinements, alternative assignments of goals to agents, and alternative resolutions of conflicts and obstacles [65].

SAS Case Study. The SAS design team identified the design decisions, options and optimisation goals shown in Figure 3. Following an approach similar to that used in many goal-oriented decision models [3, 5, 24, 32, 64], they defined the goal evaluation functions as the sum of the contributions of each option composing an architecture:

$$v(g, a, \omega) = \sum_{d \in D} contrib(g, a(d))$$

where $contrib(g, o)$ are model parameters denoting the contribution

[1]Throughout the paper, we use the term *option* to denote an alternative for a design decision and the term *alternative* to denote an alternative candidate architecture in the design space A.

of option o to goal g. For example, $contrib(BatteryUsage, GPS)$ denotes the contribution of GPS to battery usage. Since the model has 25 options and 7 goals, we have 25×7 (175) parameters.

Like all models, this model is imperfect. For example, evaluating the response time of an architecture by summing up the response time of its individual component is a basic performance model that will only give a rough approximation of an architecture response time. Evaluating the reliability of an architecture by summing the reliability of its components is most likely to be an inaccurate measure of the true reliability. Another significant problem with this model is that the goals have no clear definition. For example, what is meant by reliability and battery usage?. Similarly, the levels of contribution of each option to each goal have no clear semantics (for example, what does the contribution of the GPS to battery usage, $contrib(BatteryUsage, GPS)$, actually measure?).

In order to separate issues concerning the validity of the SAS decision model from discussions concerning the benefits of alternative decision support methods, we temporarily assume this MOADM to be valid. We revisit this assumption after having compared the two decision methods on the same model.

3.2 Defining the Cost-Benefit Model

Multi-objective decision problems increase in difficulty as the number of objectives increases [34]. Since a MOADM could have a large number of optimisation goals, one way to simplify the problem is to convert the MOADM into a simpler cost-benefit decision model [42, 49]. The cost-benefit model allows software architects to relate design decisions and levels of goal satisfaction to financial goals of direct interest to the project clients and stakeholders.

The set of alternatives of the cost-benefit decision model is the set of candidate architectures in A satisfying the constraints in C. Software architects, in collaboration with project stakeholders, define the cost and benefit functions. The parameters of the cost-benefit decision model include the parameters Ω of the architecture decision model plus additional parameters involved in the definition of the cost and benefit functions. The cost function would typically include software development, deployment, operation and maintenance costs but possibly also other costs incurred in the application domain such as salary, material, legal, environmental, and reputation costs. The benefit function would model estimated financial values associated with achieved levels of goal attainment.

A problem with many cost-benefit models is that they exclude from their equations costs and benefits that are perceived to be too hard to quantify and measure. For example, they omit the cost and benefit related to security, usability, company reputation, *etc.* To be useful, cost-benefit models should include the hard-to-measure factors that are important to the decision so that their uncertainty can be assessed and analysed instead of being ignored. Systematic methods for transforming vague qualitative goals into meaningful measurable objectives exist and have been used successfully in many industrial projects [1, 31, 41].

Many other projects however ignore these methods. A popular alternative is to compute for each alternative a utility score defined as the weighted sum of the stakeholders' preferences for each goal:

$$U(a,\omega) = \sum_{g \in G} w(g) \times Pref_g(v(g,a,\omega))$$

where the goal weights $w(g)$ and preferences functions $Pref_g(x)$ are elicited from stakeholders using appropriate techniques [61]. The goal preference values $Pref_g(x)$ are real numbers in $[0,1]$ denoting the level of preference stakeholders associate with a value x for goal g. A preference of 1 denotes the highest possible stakeholders' satisfaction, a preference of 0 denotes the worst. For example, if g is the response time of a web application, a preference of 1 may be given to an average response time of 1 second or less and of 0 to an average response time of 10 seconds or above. Preference functions are often constructed as linear or s-shape functions between the goal attainments corresponding to the lowest and highest preference [56]. This approach, or a close variant, is found in many requirements engineering methods [3, 5, 24, 32, 64].

An advantage of defining utility as a weighted sums of goal preferences is that it is extremely easy to apply. Its biggest inconvenience, however, is that the utility scores correspond to no physical characteristics in the application domain making them hard to interpret and impossible to validate empirically. In other words, the utility functions are not falsifiable [55]. In contrast, in other domains, *e.g.* in healthcare economics, utility functions are not restricted to weighted sums and they denote domain-specific measures — such as the quality-adjusted life year — making it possible to refute and improve them based on empirical evidences [7].

When a utility function exists, whether the utility is falsifiable or not, it is possible to convert a utility score into financial units using a *willingness-to-pay* ratio K such that the benefit of an alternative is the product of its utility and K [7]: $benefit(a,\omega) = K \times U(a, omega)$ This approach allows us to apply our statistical cost-benefit analysis method on any requirements and architecture models developed using a utility-based approach.

SAS Case Study. The GuideArch method uses the equivalent of a weighted sum approach to define a utility score for each candidate architecture[2]. The goal preferences are defined as linear functions where the preference 0 and 1 are associated to the lowest and highest possible values for that goal among all candidate architectures and all possible parameters' values. Therefore, instead of defining the goal preference functions in terms of stakeholder's preferences, the GuideArch model views these functions as normalisation functions expressing the percentage of goal attainment relative to the highest attainment achievable within the model. The SAS model utility score mixes both cost and benefit factors. For our experiment, we have thus assumed this utility score corresponds to the net benefit of our cost-benefit model, *i.e.* $NB(a,\omega) = U(a,\omega)$, without distinguishing the cost and benefit parts of the utility function.

3.3 Defining Design Decision Risks

Software design decisions should take into consideration the risks associated to each candidate architecture.

In a cost-benefit model, these risks can be measured using the loss probability and probable loss magnitude introduced in Section 2. Decision makers can introduce additional risk measures related to net benefits, for example measuring the probability that the net benefit or return-on-investment (*i.e.* the ratio between net benefit and cost) are below some thresholds.

In addition to risk measures related to net benefits, software architects may be interested in risks relative to the goals of the multi-objective architecture decision model:

Goal Failure Risks. The risk for an architecture a to fail to satisfy a goal g, noted $GRisk(g,a)$ is the probability that a fails to achieve some minimum level of goal attainment:

$$GRisk(g,a) = P(v(g,a,\omega) < must(g))$$

[2]The GuideArch approach assigns to each architecture a a score $s(a)$ to be minimized rather than maximised. To facilitate exposition and relation to other work, we convert the GuideArch score to a utility score to be maximised. We reproduced the GuideArch method on the SAS case study and verified our change did not affect the results; our findings are available at `www.cs.ucl.ac.uk/staff/e.letier/sdda`.

where $must(g)$ is the level of goal attainment below which stakeholders would consider the goal to be unrealized. This definition assumes g is to be maximised; a symmetric definition can be given for goals to be minimized. Eliciting the $must(g)$ values is part of many requirements engineering methods [31, 47, 56]).

Project Failure Risk. The risk for an architecture a to fail the whole project, noted $PRisk(a)$ is defined as the risk of failing to satisfy at least one of its goals. If the goals are statistically independent, we have

$$PRisk(a) = 1 - \prod_{g \in G} (1 - GRisk(g,a)).$$

The project failure risk is defined with respect to goals are defined in the multi-objective architecture decision model. These goals may include concerns related to development costs and schedule.

SAS Case Study. The original SAS model has no definition of risk and does not specify *must* values for any of its goals. We thus decided to define the $must(g)$ values relative to the goal level attainment of a baseline architecture whose goal attainments would be equal to those of the existing system. The new system has to be at least as good as the current system on all goals, otherwise the project would be a failure. We selected as baseline the architecture with the lowest expected net benefit from among the top 5%.

3.4 Eliciting Parameters Values

The following step consists in eliciting probability distributions (or single value in case a parameter is known with certainty) for all parameters in the architecture and cost benefit decision models. As mentioned in Section 2, simple, reliable methods exist for performing this elicitation [52].

SAS Case Study. The SAS design team elicited uncertainty for all 175 model parameters through a three-point estimation method that consist in eliciting for each parameter its most likely, lowest and highest values. They interpreted these three points estimates as triangular fuzzy value functions which are equivalent to triangular probability distributions. They also elicited point-based values for each of the 7 goal weights parameters (unlike our approach, GuideArch does not allow these weights to be uncertain).

3.5 Shortlisting Candidate Architectures

The next step consists in shortlisting candidate architectures to be presented to software architects for the final decision and for computing expected information value.

For this step, software architects have to decide what shortlisting criteria to use. The default is to shortlist candidate architectures that maximise expected net benefit and minimise project failure risk. Software architects may, however, select other risk-related criteria such as the probabilities that the project costs and schedule exceed some threshold, or that the loss probability or probable loss magnitude do. Software architects may select any number of criteria. However, keeping the number of criteria below 3 facilities the generation and visualisation of the shortlist.

Architects may also specify, for each criteria, a resolution margin to resist specious differentiation when comparing alternatives. For example, setting the resolution margins for financial objectives, such as expected net benefit, to £10,000 causes the shortlisting process to ignore differences of less then £10,000 when comparing candidate architectures net benefit. These margins make our shortlisting process robust against statistical error in the MC simulation and modelling error due to simplifications in the model equations. Shortlisting candidate architectures based on strict Pareto-optimality without our resolution margin can cause a priori rejection of a candidate architecture due to insignificant differences in objective attainment.

Our tool then computes the shortlist as the set of Pareto-optimal candidature architectures for the chosen criteria and resolution margins. More precisely, a candidate architecture a is shortlisted if there is no other candidate architecture a' that outperforms a by the resolution margins on *all* criteria. If the MOADM includes a non-empty set C of dependency constraints between design decisions, any architecture that violates these constraints is automatically excluded. Our shortlisting approach is an extension of the standard notion of Pareto-optimality [34] used to deal with optimisation problems involving uncertainty. In the objective space, the outcomes of each candidate architecture for each criteria forms a Pareto-optimal *strip*, or a Pareto-optimal front with margins.

Our implementation identifies the Pareto-optimal alternatives through an exhaustive exploration of the design space. It first computes the $\widehat{NB}$ matrix for the full design space using MC simulation then uses a classic algorithm for extracting Pareto-optimal sets [46] that we have extended to deal with resolution margins. Our implementation is in `R`, an interpreted programming language for statistical computing.

For the SAS model, on a standard laptop, the MC simulations of all 6912 alternatives takes around 5 minutes (for a MC simulation with 10^4 scenarios) and the identification of the Pareto-optimal strip less than a second. Other industrial architecture decision problems have a design space whose size is similar or smaller to that of the SAS [9, 43, 44]. For example, the application of CBAM to NASA Earth Observation Core System (ECS) [43] involves 10 binary decisions (thus 1024 alternative architectures against 6912 for the SAS). Our exhaustive search approach is thus likely to be applicable to most architecture decision problems.

The scalability bottleneck of our approach is more likely to be related to the elaboration of the decision models (steps 1 and 2) and the number of parameters to be elicited from stakeholders (step 3) than to the automated shortlisting step. If, however, a need to increase the performance and scalability of our shortlisting technique appears, one could port our implementation to a faster complied programming language and use evolutionary algorithms commonly used in search-based software engineering [33] such as NSGA2 [15] to deal with much larger design spaces (but at the cost of losing the guarantee of finding the true Pareto-optimal strip).

SAS Case Study. Figure 4 shows the Pareto-optimal strip for the SAS candidate architectures evaluated with respect to expected net benefit and project failure risk. The resolution margins for each criteria are set at 0.1 and 1%, respectively. The red crosses show the 9 architectures shortlisted by our approach, the blue squares the top 10 architectures of the GuideArch approach, and the grey circles all other candidate architectures. In our shortlist, 5 out of 9 candidate architectures are in the Pareto-strip but not on the Pareto-front; they would have groundlessly been excluded from the shortlist if we had followed the traditional approach of retaining solutions in the Pareto-optimal front only.

We observe important differences between our shortlist and top 10 architectures GuideArch identifies: our shortlists identifies candidate architectures with slightly higher expected net benefit and much lower project risk than GuideArch's top 10 architectures. We explain the difference between the two shortlists as follows. GuideArch did not consider project failure risk as we defined it in Section 3.3 (or any other risk) in their architecture evaluations. It is therefore not surprising that its top 10 architectures perform weakly with respect to this criterion. Instead of evaluating criteria against their expected net benefit (or equivalently their utility score) and some measure of risk, GuideArch ranks candidate architectures according to a single criterion corresponding to an uncertainty-adjusted score defined as the weighted sum of an architecture's pessimistic, most

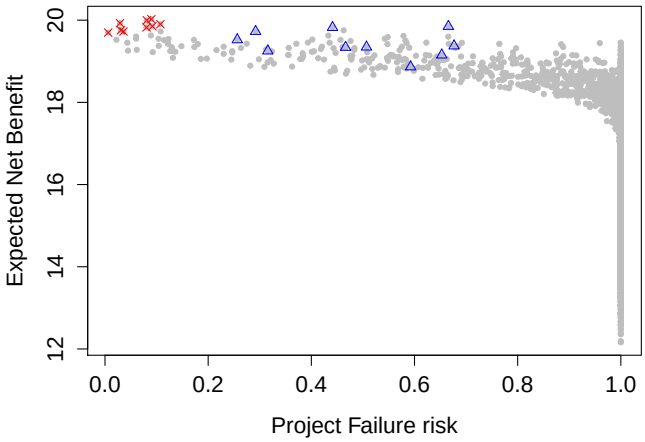

Figure 4: Comparing our shortlisted architectures (red crosses) against GuideArch top 10 (blue triangles). The grey circles denote all other candidate architectures.

Open Decisions	Options	
File Sharing	OpenIntents	In house
Chat	XMPP (Open File)	In house
Connectivity	3G on Nexus 1	3G on Droid
Architectural Pattern	Facade	Psuh-based

Closed Decisions	Option
Location Finding	Radio
Hardware Platform	Nexus 1
Report Syncing	Explicit
Map Access	Preloaded (ESRI)
Database	MySQL
Data Exchange Format	Unformatted Data

Figure 5: Open and closed decisions in our shortlisted architectures.

likely, and optimistic net benefit in fuzzy logic. The weights in the uncertainty-adjusted score capture the importance decision makers give to pessimistic, most likely, and optimistic outcomes. In other words, GuideArch scores architectures by taking into account the most likely net benefits (in probabilistic terms, the mode of the distribution) and what its authors call the positive and negative consequences of uncertainty. In our probabilistic approach, both types of consequences are already taken into account by computing the *expected* net benefit. We argue our shortlisting approach has two advantages over the GuideArch uncertainty-adjusted scores: (1) it informs decision makers of both expected net benefit and risks; and (2) it does not require decision makers to specify uncertainty-adjusting weights whose impacts on the architectures ranking are difficult to interpret.

3.6 Open and Closed Design Decisions

Shortlisting a set of candidate architectures may conclude a set of design decisions. A design decision is closed if all shortlisted architectures agree on the option to be selected for this decision; a design decision is open if the shortlisted architecture contains alternative options for that option. Presenting the open and closed design decisions gives decision makers a useful view of the shortlisted architectures. If the shortlist is large, it can also be organised into clusters based on design decisions similarities [66].

SAS Case Study. Figure 5 shows the open and closed design decisions in our shortlisted candidate architectures.

	$EVPPI/EVTPI$	$\Delta_{PPIP/TPI}(Risk)$
Ramp up time (1)	10%	5%
Battery Usage (1)	10%	5%
Ramp up time (14)	10%	5%
Battery Usage (20)	10%	4%
Ramp up time(11)	10%	4%
Ramp up time(20)	10%	4%
Battery Usage (11)	10%	5%
Development Time (20)	9%	5%
Development Time (1)	4%	5%
Development Time (14)	3%	5%
Development Time (11)	3%	5%

Figure 6: Expected Value of Partial Perfect Information.

3.7 Computing Information Value

The last step consists in computing the expected value of perfect information and its impact on risks. The expected value of *total* perfect information and its impact on risk, *EVTPI* and *ERITPI*, give upper bounds on the value that additional information could bring to the decision. If *EVTPI* is small and the impact on risk low, there is little value in reducing model parameters uncertainty. The expected value of *partial* perfect information about a single model parameter Θ and its impact on risk, $EVPPI(\Theta)$ and $ERIPPI(\Theta)$, help software architects to distinguish model parameters with high and low expected information value. We also found it useful to measure the expected value of partial perfect information about the level of attainment of each goal and its impact on risk, $EVPPI(v(g,a))$ and $ERIPPI(v(g,a))$. This gives software architects a mean of separating high and low information value at the levels of goals instead of individual parameters which can be too numerous (the SAS model has 175 parameters) and fine-grained.

To ease computations of expected information values, we limit the alternatives to those in the shortlist. In our case study, this reduces the $\widehat{NB}$ matrix from which *EVTPI* and *EVPPI* are computed from a size of 6912 by 10^4 (the number of alternatives by the number of simulation scenarios) to a size of 9 by 10^4.

One should be careful in interpreting *EVTPI* and *EVPPI* values to remember that their accuracy is conditional on the validity of the decision model. They only measure the value of reducing uncertainty about model parameters, not about the model equations. We come back to this issue below.

SAS Case Study. Using the shortlisted architectures identified in Section 3.5 and $\widehat{NB}$ matrix for those architectures, we compute that *EVTPI* is 0.05 which represents only 0.25% of the highest expected net benefit. EIRTPI is 9% which is the full project failure risk of the highest benefit architecture in our shortlist. This means that the impact of perfect information is to reduce project failure risk to zero. Figure 6 shows all non-zero EVPPI for all goals and architectures. Since these EVPPI are small, the table shows the ratio of EVPPI to EVTPI instead of absolute values. The ramp up time and battery usage of 4 of the 9 shortlisted architectures are shown to have, in relative terms, much higher information value than other goals and architectures. However, in absolute terms, these values remain low.

In order to experiment with the use of EVTPI and EVPPI, we have artificially extended uncertainty in the SAS model and observed the effect on EVTPI and EVPPI. We have, for example, given uncertainty to the goal weights in the definition of the utility function. We have assumed that the SAS design team is likely to have overestimated the goal weights and have therefore replaced their constant value by a triangular distribution of parameters $(0, w(g), w(g))$ where $w(g)$ is the initial goal weight estimated by the SAS design

team. This distribution resulting in a linearly decreasing probability distribution function from $w(g)$ to 0. We observed that this uncertainty roughly doubled EVTPI. However, in our all experiments, EVTPI remains small. This is mostly due to the small differences in net benefit that exist among the shortlisted architectures even when most of the model parameters uncertainties are increased.

If we had confidence in the validity of the model utility function, this result would mean that, for this particular decision problem, there is no value in reducing uncertainty before deciding among the shortlisted architectures. However, we have identified important limitations in the SAS MOADM and utility models that severely question their validity, the most important problem being that these models are not falsifiable, making it impossible to validate and improve them based on empirical evidence. The project client should thus be sceptical of the choice of architecture, risk assessment, and information value generated using these models whatever decision support method is used.

In order to deal with such difficulties, it would be desirable to be able to explicitly describe and reason not only about parameters uncertainty, but also about model uncertainty (also called structural uncertainty) [17]. Requirements and architecture decision problems would especially benefit from this capability. It would enable an incremental approach where software architects could start from an inexpensive, coarse-grained decision model with large uncertainty, then use expected information value about model uncertainty to decide whether and where to reduce uncertainty by refining parts of the model. They could for example start with a coarse-grained software performance model similar to the one used in the SAS case study, estimate their uncertainty about the model error (the deviation between its predicted performance and the software's actual performance) and compute the expected value of perfect information about this error to decide whether to refine this model into a more fine-grained performance model. We have started exploring how to extend our method to deal with model uncertainty by converting it to parameter uncertainty, but the approach is still tentative and our method does not currently supports this.

4. EVALUATION AND FUTURE WORK

Evaluating decision support methods is hard. Often, authors argue that their method is systematic, liked by its users, triggers useful discussions and generates insights into the decision problem [21, 34, 49]. None of these claims, however, consider whether a decision method is correct and produces better outcomes than another method, or even than no method at all (*i.e.*. decisions based on intuition alone). The popular AHP method [57], for example, is criticised by decision experts for its mathematical flaws [8, 10, 53] and lack of evidence that it leads to better decisions than intuition alone [40].

Evaluation of software engineering decision methods should go beyond vague claims of usefulness. In this section, we propose to evaluate software engineering decision support methods according to their correctness, performance and scalability, applicability, and cost-effectiveness. Inspired by an adaptation of Moslow's pyramid of human needs to software quality [2], we visualize these criteria in a pyramid where the lower-level criteria are necessary foundations for higher-level ones. We discuss the extent to which we can claim our method meets these criteria and outline a roadmap of future research to extend our evaluation and improve our method against those criteria.

1. Correctness. The first level is to establish what correctness properties can be claimed of the method. One must distinguish correctness of the decision *method* from correctness of the decision *models* to which the method is applied. Our method is correct in the sense that its produces correct estimations of the candidate architectures expected net benefits, risks, and expected information value *assuming validity of the decision models and accuracy of the parameters' probability distributions*[3]. Not all decision methods can make this correctness claim. For example, GuideArch computes for each architecture a score that, unlike our expected net benefit and risk, makes no falsifiable predictions about the architecture and has therefore no notion of correctness.

The lack of validity and falsifiability of the decision model we used in the SAS case study is an important weakness of our evaluation. All models are wrong, but some are useful [13]. Unfortunately, today no scientific method exists to help software architects evaluate how useful a model actually is to inform decisions. As mentioned in the closing of the previous section, we intend to address this shortcoming by extending our approach to deal with model uncertainty so as the be able to estimate modelling errors, their impact on decisions, and support an incremental modelling process guided by information value analysis.

Our method assumes it is possible to elicit accurate probability distributions for all model parameters. Such elicitation can be hampered by important cognitive biases. For example, software estimations have been shown to be affected by anchoring [6]. Significant research in uncertainty elicitation has show it is possible to counter the effects of such biases using appropriate methods [52]. However, these methods have to our knowledge not yet been applied in a software engineering context and further evaluation is thus required in this area.

2. Performance and scalability. With the SAS case study, we have shown our method is fast enough to analyse a real software design decision problem whose size and complexity is similar to those of other published industrial architecture decision problems [9, 43, 44]. The manual steps of elaborating the decision models and eliciting all parameters probability distributions will most likely be the first scalability bottleneck of applying our method to more complex problems. If our automated shortlisting step becomes a bottleneck, its performance and scalability can be improved notably by using by using evolutionary search-based algorithms to reduce the number of candidate architectures to evaluate. In the near future, we intend to conduct a systematic scalability analysis [19] of the whole approach on real case studies before attempting to improve its performance.

3. Applicability. The next evaluation criteria is to show the method is applicable by its intended users (not just the method designers) in actual software engineering projects. We distinguish *technical applicability*, the extent to which the method is understandable and applicable by software architects in an ideal (fictive) project where actors do not intentionally or unintentionally game the decision making process, from *contextual applicability*, the extent to which the method is applicable in the context of real projects where the project governance, incentives, and political relations might affect the decision making process and reporting of uncertainty.

At the moment, we see no critical threats to the technical applicability of our method. Our method takes as input decision models that correspond to those already produced by other requirements engineering and architecture methods [22, 45, 47, 49]. The only other required inputs are probability distributions modelling the decision makers uncertainty about the model parameters. As mentioned earlier, simple, reliable methods exist to elicit such probability distributions [52]. Our analysis outputs need to be easily interpretable by

[3]Our approach actually computes these quantities using MC simulation which introduces bounded and measurable simulations errors [48, 54]. In our case study, with simulations of 10^5 scenarios, these errors are negligible, particularly when compared to the much wider modelling and parameter uncertainty.

decision makers. Although the concepts of risk, Pareto-optimality and information value can be misunderstood, we see no insurmountable obstacle here.

Even if the method is technically applicable, the political context and governance structure of a project may create obstacles to the accurate reporting of uncertainty and analysis of risks [40, 50]. Important research in this area will be needed to identify incentives and governance structures that are favourable to sound decision making under uncertainty.

4. Cost-effectiveness The next evaluation stage is to demonstrate the cost-effectiveness of decision analysis methods in requirements and architecture decisions. A method can be applicable without being cost-effective. Showing cost-effectiveness of a decision method dealing with uncertainty is hard. One must distinguish a good *decision* from a good *outcome*. A good decision may by the effect of chance lead to a bad outcome, and vice-versa a bad decision may also by the effect of chance lead to a good outcome. However, when analysed over many decisions, a good decision support method should on average lead to better outcomes, which for software engineering projects means higher business benefits from IT projects and less costly project failures. We believe that by setting expected benefits and risks as explicit decision criteria and by using falsifiable models that can be incrementally improved from empirical evidence, our method has a better chance of achieving these goals than other methods relying on unfalsifiable models and utility scores not clearly related to benefits and risks.

5. RELATED WORK

Most requirements and architecture decision methods ignore uncertainty and rely on point-based estimates of their models parameters [5, 22, 24, 32, 47, 64, 68]. By simply replacing point-based estimates by probability distributions, our method can be directly applied to any previous decision model because the MC simulations at the heart of the method merely consist of evaluating the point-based models on many different possible parameters values.

Our method builds on previous methods for dealing the uncertainty in software architecture decisions, notably CBAM [42, 49] and GuideArch [21].

The first two steps of our method are equivalent to CBAM's model elaboration steps. Our method differs from CBAM in that it relies on sound, reliable techniques for eliciting probability distributions; it includes explicit definition of risks against which alternatives are evaluated; it shortlists candidate architectures based on multiple objectives (*e.g.* ENB and Risk) instead of assuming a single ranking criterion; and it measures expected information value whereas CBAM uses deterministic sensitivity analysis whose limitations were described in Section 2.2. Elaborating on the first point, CBAM infers probability distributions from divergences between stakeholders' single-point estimates; this confuses consensus about the most likely value with uncertainty about the possible ranges of values.

Our method differs from GuideArch in the following ways. In step 1 and 2, our method allows decision makers to elaborate problem-specific decision models whereas GuideArch relies on fixed equations for computing a score for each candidate architecture. The GuideArch equations are not falsifiable and therefore not amenable to empirical validation. Likewise, unlike step 3 of our method, GuideArch does not allow decision makers to define domain-specific measures of risks. In step 4, we model uncertainty about parameters' values as probability distributions for which sound uncertainty elicitation techniques exist [52] whereas GuideArch uses fuzzy logic values that cannot be empirically validated and calibrated. In step 5, we allow decision makers to shortlist candidate architecture based on expected net benefit and risks whereas GuideArch ranks architecture using a single risk-adjusted score whose interpretation is problematic. Finally, GuideArch has no support for assessing the value of information. Because GuideArch does not require the elaboration of problem-specific models, it may be simpler to apply than CBAM and our approach; however, the lack of validity of the fixed equations used to score alternatives should raise concerns regarding the validity of the rankings it produces.

Our decision support method deals with design time *knowledge* uncertainty and should not be confused with the large body of software engineering research dealing with run-time *physical* uncertainty (*e.g.* [30, 35, 36, 47]). Philosophers and statisticians use the terms epistemic and aleatory uncertainty, respectively [52]. A probabilistic transition system may for example describe variations in the response time of a web service as an exponential distribution with a mean λ. This models a run-time physical uncertainty. Such probabilistic model could be part of a decision model where the mean λ is an uncertain model parameter. The decision makers' uncertainty about λ is a knowledge uncertainty.

Other software engineering research streams are concerned with uncertainty during the elaboration of partial models [23] and uncertainty in requirements definitions for adaptive systems [67]. These are different concerns and meanings of uncertainty than those studied in this paper.

Graphical decision-theoretic models [18] and Bayesian networks [28] provide general tools for decision making under uncertainty. They have supported software decisions regarding development resources, costs, and safety risks [26, 27] but not requirements and architecture decisions. We did not used these tools to support our method because they deal with discrete variables only; their use would have required transforming our continuous variables such as cost, benefit and goal attainment levels into discrete variables.

Boehm's seminal book on software engineering economics devotes a chapter to statistical decision theory and the value of information [11]. The chapter illustrates the expected information value on a simple example of deciding between two alternative development strategies. To our knowledge, this is the only reference to information value in the software engineering literature, including in Boehm's subsequent work. This concept thus appears to have been forgotten by our community.

Software cost estimation methods [4, 12, 26, 60] could be used to provide inputs to our decision method. Many already rely on statistical and Bayesian methods to provide cost estimates; they could easily generate cost estimates in the form of probability distributions instead of point-based estimates.

6. CONCLUSION

Requirements and architecture decisions are essentially decisions under uncertainty. We have argued that modelling uncertainty and mathematically analysing its consequences leads to better decisions than either hiding uncertainty behind point-based estimates or treating uncertainty qualitatively as an inherently uncontrollable aspect of software development. We believe that statistical decision analysis provide the right set of tools to manage uncertainty in complex requirements and architecture decisions. These tools may be useful to other areas of software engineering, *e.g.* testing, where critical decisions must be made by analysing risks arising out of incomplete knowledge. In future work, we intend to validate and refine our method on a series of industrial case studies and address the problem of reasoning about model uncertainty.

7. REFERENCES

[1] G. Adzic. *Impact Mapping: Making a big impact with software products and projects*. Provoking Thoughts, 2012.

[2] G. Adzic. Redefining software quality. `http://gojko.net/2012/05/08/redefining-software-quality`, 2012. Last Accessed: 28 Feb 2014.

[3] Y. Akao. *Quality function deployment: integrating customer requirements into product design*. Productivity Press, 1990.

[4] A. J. Albrecht. Measuring application development productivity. In *Proceedings of the Joint SHARE/GUIDE/IBM Application Development Symposium*, volume 10, pages 83–92, 1979.

[5] D. Amyot, S. Ghanavati, J. Horkoff, G. Mussbacher, L. Peyton, and E. Yu. Evaluating goal models within the goal-oriented requirement language. *International Journal of Intelligent Systems*, 25(8):841–877, 2010.

[6] J. Aranda and S. Easterbrook. Anchoring and adjustment in software estimation. In *Proceedings of the 10th European Software Engineering Conference Held Jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ESEC/FSE-13, pages 346–355. ACM, 2005.

[7] G. Baio. *Bayesian Methods in Health Economics*. CRC Press, 2012.

[8] C. A. Bana e Costa and J.-C. Vansnick. A critical analysis of the eigenvalue method used to derive priorities in AHP. *European Journal of Operational Research*, 187(3):1422–1428, 2008.

[9] L. Bass, P. Clements, and R. Kazman. *Software architecture in practice*. Addison-Wesley Professional, 2003.

[10] V. Belton and T. Gear. On a short-coming of saaty's method of analytic hierarchies. *Omega*, 11(3):228–230, 1983.

[11] B. W. Boehm. *Software Engineering Economics*. Prentice Hall, Englewood Cliffs, NJ, 1981.

[12] B. W. Boehm, R. Madachy, B. Steece, et al. *Software Cost Estimation with Cocomo II*. Prentice Hall PTR, 2000.

[13] G. E. Box and N. R. Draper. *Empirical model-building and response surfaces*. John Wiley & Sons, 1987.

[14] M. Cantor. Calculating and improving ROI in software and system programs. *Commun. ACM*, 54(9):121–130, Sept. 2011.

[15] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-II. *Lecture notes in computer science*, 1917:849–858, 2000.

[16] M. Denne and J. Cleland-Huang. The incremental funding method: Data-driven software development. *IEEE Software*, 21(3):39–47, 2004.

[17] D. Draper. Assessment and propagation of model uncertainty. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 45–97, 1995.

[18] M. J. Druzdzel. Smile: Structural modeling, inference, and learning engine and genie: a development environment for graphical decision-theoretic models. In *AAAI/IAAI*, pages 902–903, 1999.

[19] L. Duboc, E. Letier, and D. S. Rosenblum. Systematic elaboration of scalability requirements through goal-obstacle analysis. *IEEE Transactions on Software Engineering*, 39(1):119–140, 2013.

[20] N. Esfahani and S. Malek. Guided exploration of the architectural solution space in the face of uncertainty. Technical report, 2011.

[21] N. Esfahani, S. Malek, and K. Razavi. GuideArch: guiding the exploration of architectural solution space under uncertainty. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 43–52. IEEE Press, 2013.

[22] D. Falessi, G. Cantone, R. Kazman, and P. Kruchten. Decision-making techniques for software architecture design: A comparative survey. *ACM Computing Surveys (CSUR)*, 43(4):33, 2011.

[23] M. Famelis, R. Salay, and M. Chechik. Partial models: Towards modeling and reasoning with uncertainty. In *34th International Conference on Software Engineering (ICSE 2012)*, pages 573–583. IEEE, 2012.

[24] M. S. Feather and S. L. Cornford. Quantitative risk-based requirements reasoning. *Requirements Engineering*, 8(4):248–265, 2003.

[25] J. C. Felli, G. B. Hazen, P. D, and P. D. Sensitivity analysis and expected value of perfect information. *Medical Decision Making*, 18:95–109, 1997.

[26] N. Fenton, W. Marsh, M. Neil, P. Cates, S. Forey, and M. Tailor. Making resource decisions for software projects. In *26th International Conference on Software Engineering (ICSE 2004)*, pages 397–406. IEEE, 2004.

[27] N. Fenton and M. Neil. Making decisions: using bayesian nets and MCDA. *Knowledge-Based Systems*, 14(7):307–325, 2001.

[28] N. Fenton and M. Neil. *Risk Assessment and Decision Analysis with Bayesian Networks*. CRC Press, 2012.

[29] B. Flyvbjerg and A. Budzier. Why your IT project may be riskier than you think. *Harvard Business Review*, 89(9):23–25, 2011.

[30] C. Ghezzi, L. S. Pinto, P. Spoletini, and G. Tamburrelli. Managing non-functional uncertainty via model-driven adaptivity. In *35th International Conference on Software Engineering (ICSE 2013)*, pages 33–42. IEEE Press, 2013.

[31] T. Gilb. Competitive engineering: : A handbook for systems engineering, requirements engineering, and software engineering using Planguage. *Butterworth-Heinemann Ltd*, 2005.

[32] P. Giorgini, J. Mylopoulos, E. Nicchiarelli, and R. Sebastiani. Reasoning with goal models. In *Conceptual Modeling – ER 2002*, pages 167–181. Springer, 2002.

[33] M. Harman, S. A. Mansouri, and Y. Zhang. Search-based software engineering: Trends, techniques and applications. *ACM Computing Surveys*, 45(1):11, 2012.

[34] M. Harman, P. McMinn, J. T. de Souza, and S. Yoo. Search based software engineering: Techniques, taxonomy, tutorial. In *Empirical Software Engineering and Verification*, pages 1–59. Springer, 2012.

[35] W. Heaven and E. Letier. Simulating and optimising design decisions in quantitative goal models. In *19th IEEE International Requirements Engineering Conference (RE 2011)*, pages 79–88. IEEE, 2011.

[36] A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker. PRISM: A tool for automatic verification of probabilistic systems. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 441–444. Springer, 2006.

[37] R. Howard. Information value theory. *IEEE Transactions on Systems Science and Cybernetics*, 2(1):22–26, 1966.

[38] R. A. Howard. *Readings on the principles and applications of decision analysis*, volume 1. Strategic Decisions Group, 1983.

[39] D. Hubbard. The IT measurement inversion. *CIO Enterprise Magazine*, 1999.

[40] D. Hubbard. *The Failure of Risk Management: Why It's*

Broken and How to Fix It. Wiley, 2009.

[41] D. Hubbard. *How to measure anything: Finding the value of intangibles in business*. Wiley, 2010.

[42] R. Kazman, J. Asundi, and M. Klein. Quantifying the costs and benefits of architectural decisions. In *23rd International Conference on Software Engineering (ICSE 2001)*, pages 297–306. IEEE Computer Society, 2001.

[43] R. Kazman, J. Asundi, and M. Klien. Making architecture design decisions: An economic approach. Technical report, DTIC Document, 2002.

[44] R. Kazman, M. Barbacci, M. Klein, S. Jeromy Carriere, and S. G. Woods. Experience with performing architecture tradeoff analysis. In *21st International Conference on Software Engineering (ICSE 1999)*, pages 54–63. IEEE, 1999.

[45] R. Kazman, M. Klein, M. Barbacci, T. Longstaff, H. Lipson, and J. Carriere. The architecture tradeoff analysis method. In *Fourth IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'98)*, pages 68–78. IEEE, 1998.

[46] H.-T. Kung, F. Luccio, and F. P. Preparata. On finding the maxima of a set of vectors. *Journal of the ACM*, 22(4):469–476, 1975.

[47] E. Letier and A. van Lamsweerde. Reasoning about partial goal satisfaction for requirements and design engineering. In *12th International Symposium on the Foundation of Software Engineering (FSE 2004)*, volume 29, pages 53–62. ACM, 2004.

[48] D. Lunn, C. Jackson, D. J. Spiegelhalter, N. Best, and A. Thomas. *The BUGS book: A practical introduction to Bayesian analysis*, volume 98. CRC Press, 2012.

[49] M. Moore, R. Kazman, M. Klein, and J. Asundi. Quantifying the value of architecture design decisions: lessons from the field. In *25th International Conference on Software Engineering (ICSE 2003)*, pages 557–562, 2003.

[50] National Audit Office. Over-optimism in government projects, 2013.

[51] B. Nuseibeh. Weaving together requirements and architectures. *IEEE Computer*, 34(3):115–119, 2001.

[52] A. O'Hagan, C. Buck, A. Daneshkhah, J. Eiser, P. Garthwaite, D. Jenkinson, J. Oakley, and T. Rakow. *Uncertain Judgements: Eliciting Experts' Probabilities*. Statistics in Practice. Wiley, 2006.

[53] J. Pérez, J. L. Jimeno, and E. Mokotoff. Another potential shortcoming of AHP. *Top*, 14(1):99–111, 2006.

[54] M. Plummer. Jags: A program for analysis of bayesian graphical models using Gibbs sampling. In *Proceedings of the 3rd International Workshop on Distributed Statistical Computing (DSC 2003). March*, pages 20–22, 2003.

[55] K. Popper. *The logic of scientific discovery*. Routledge, 1959.

[56] B. Regnell, R. B. Svensson, and T. Olsson. Supporting roadmapping of quality requirements. *IEEE Software*, 25(2):42–47, 2008.

[57] T. L. Saaty. How to make a decision: the analytic hierarchy process. *European journal of operational research*, 48(1):9–26, 1990.

[58] M. Sadatsafavi, N. Bansback, Z. Zafari, M. Najafzadeh, and C. Marra. Need for speed: an efficient algorithm for calculation of single-parameter expected value of partial perfect information. *Value in Health*, 2013.

[59] P.-Y. Schobbens, P. Heymans, and J.-C. Trigaux. Feature diagrams: A survey and a formal semantics. In *14th IEEE International Requirements Engineering Conference (RE 2006)*, pages 139–148. IEEE, 2006.

[60] M. Shepperd. Software project economics: a roadmap. In *Future of Software Engineering, 2007. FOSE'07*, pages 304–315. IEEE, 2007.

[61] T. J. Stewart. Dealing with uncertainties in MCDA. In *Multiple criteria decision analysis: State of the art surveys*, pages 445–466. Springer, 2005.

[62] M. Svahnberg, C. Wohlin, L. Lundberg, and M. Mattsson. A quality-driven decision-support method for identifying software architecture candidates. *International Journal of Software Engineering and Knowledge Engineering*, 13(5):547–573, 2003.

[63] R. N. Taylor, N. Medvidovic, and E. M. Dashofy. *Software architecture: foundations, theory, and practice*. Wiley, 2009.

[64] A. van Lamsweerde. Reasoning about alternative requirements options. In *Conceptual Modeling: Foundations and Applications*, pages 380–397. Springer, 2009.

[65] A. van Lamsweerde. *Requirements engineering: from system goals to UML models to software specifications*. Wiley, 2009.

[66] V. Veerappa and E. Letier. Understanding clusters of optimal solutions in multi-objective decision problems. In *19th IEEE International Requirements Engineering Conference (RE 2011)*, pages 89–98. IEEE, 2011.

[67] J. Whittle, P. Sawyer, N. Bencomo, B. H. Cheng, and J.-M. Bruel. Relax: Incorporating uncertainty into the specification of self-adaptive systems. In *17th IEEE International Requirements Engineering Conference (RE 2009)*, pages 79–88. IEEE, 2009.

[68] Y. Zhang, A. Finkelstein, and M. Harman. Search based requirements optimisation: Existing work and challenges. In *Requirements Engineering: Foundation for Software Quality*, pages 88–94. Springer, 2008.

[69] Y. Zhang and M. Harman. Search based optimization of requirements interaction management. In *Second International Symposium on Search Based Software Engineering (SSBSE 2010)*, pages 47–56, 2010.