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 risks. Yet software architects lack support for evaluating uncertainty, its impact on risk, and the value of reducing uncertainty, *e.g.* through additional elicitation, modelling, or prototyping, 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 mobile 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.

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 decision questions 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 loss, disaster, or other undesirable events. 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. As usual in engineering, risk is characterized by the likelihood and severity of the loss, disaster or undesirable events [35]. In software architectural 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 [5]. 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 architectural decisions, i.e. decisions about the functionality the software should provide, the quality requirements it should satisfy, its organisation into components and interfaces, and its deployment topology. We assume stakeholders' goals and the alternatives have been identified using appropriate requirements engineering and software architecture methods [57, 40, 55, 38]. 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 [37, 42, 54, 16] 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 [33]. Decision analysis is used notably in the health care domain to inform decisions about the cost-effectiveness of new medical treatments based on the results of clinical trials [6]. There are exceptional uses of these methods in the context of IT investment decisions [36, 9], 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 contributions are the following:

1. It shows how decision analysis methods can be used for requirements and architecture decisions and the benefits these methods bring over other, previously proposed approaches (Section 3);
2. It defines a set of novel decision risk metrics tailored for requirements and architecture decision problems (Section 3.3);
3. It introduces the concept of Pareto-optimal *strip*, a generalisation of a Pareto-optimal front, designed to resist modelling and measurement errors, that we use to shortlist candidate solutions during decision making (Section 3.5);
4. It extends the concept of value of information, traditionally defined in terms impact of additional information on expected outcomes only, by considering how additional information reduces risk (Section 2.3).

We have developed a tool supporting our approach and have applied it to data from a real system from the literature [16]. 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 software architecture decision problems, we first consider a simpler decision problem where one alternative must be selected from among a set of alternatives based on their costs and benefits. Such problems assume a model exists that calculates the costs and benefits of all alternatives in a common unit, which is usually monetary (*e.g.* Pound, Euro, Dollar, Yen or Rupee) [6]:

Definition 1. 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 needs to decide whether to replace an ageing Computer-Aided Despatch (CAD) application with a new system. The set of alternatives is thus $A = \{\text{legacy}, \text{new}\}$. 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 to 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 [9, 11]. The model parameters are the independent 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. The decision is difficult because the firm is losing market shares due to its reliance on an ageing CAD, but developing a new system requires a high investment with uncertain benefits.

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 the model parameters. The obvious problem with this approach is that it ignores the often large uncertainty about these parameters values. Uncertainty about cost and benefit exists but it is not shown. In a statistical CBA, uncertainty about the model parameters are modelled explicitly as probability distributions and they are used to compute a probability distribution for the net benefit of each alternative.

Simple, effective methods exist for eliciting the model parameters' probability distributions from decision makers [44]. These methods are founded on sound mathematical foundations and significant empirical studies of how uncertainty can be reliably elicited from humans. They have notably shown how estimation skills can be significantly improved through calibration exercises. 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 distribution have been estimated, one needs to compute the probability distributions for cost, benefit and net benefit of each alternatives. It is generally not possible to compute these probability distributions analytically because the model equations and parameter's probability distributions can be arbitrarily complicated. Good approximations of the probability distributions for cost, benefit and net benefit can however be computed through Monte-Carlo (MC) simulations. The underlying principle is to sample a large number of simulation scenarios generated by model parameter values drawn from their probability distributions and uses 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 simulation scenarios and N is the number of alternatives in A. The element $\widehat{NB}[i,j]$ denotes the net benefit obtained 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:

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

Example Figure 1 shows the results of a statistical CBA analysis for our illustrative example. For simplicity, we assume that the probability distribution for the cost and benefit of each alternative have been elicited directly from the decision makers. 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 analysis, these risks would not have been

	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.

quantified and they would, most likely, have been underestimated if not entirely ignored.

2.2 The Value of Information

If decision makers can obtain additional information about the alternatives they face, obtaining this information may reduce their uncertainty which may, in turn, allow them to make a better decision resulting in higher net benefit. The increase in net benefit brought about by this new information measures the value of this additional information [32].

Knowing the value of information helps decision makers decide what information to seek: they should prioritize high value information and avoid paying more for information than it is worth. Computing the value of information can provide important and unexpected information to decision makers. After applying information value theory to 20 IT project business cases (each having between 40 to 80 variables), Hubbard 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 [36]. The contrast between the second and third observations constitutes what Hubbard has called the IT measurement inversion paradox [34]. He cites the large effort spent by one of its client on function point analysis [3] — a popular software development productivity and cost estimation method — as an example of measurement with very low information value because its cost estimation weren't more accurate or precise than the project managers' initial estimates.

The 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 definitions apply to any other outcome, *e.g.* maximising software reliability. Information value as the same unit 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:

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

Using a standard notation, $E[x]$ denotes the expectation of a random variable x. When decision makers know the exact value ω of all model parameters, their gain in net benefit is the difference between their maximal net benefit knowing the parameters values to be ω, *i.e.* the first term of the expectation $\max_{a \in A} NB(a,\omega)$ and their maximal net benefit not knowing the exact values of Ω, *i.e.* the second term $\max_{a \in A} E[NB(a,\omega)]$. In other words, with perfect information, decision makers can select an alternative with maximal *actual* net benefit for the known values ω of the model parameters. Without perfect information, they can only select an alternative with maximal *expected* net benefit over the probability distribution of Ω. The EVTPI is the expectation of this gain over the probability distribution of Ω, *i.e.* over all possible, concrete values ω the model parameters could take. The *EVTPI* is always positive zero. It can be estimated from the output $\widehat{NB}$ of a MC simulation:

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

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

Figure 2: A Monte-Carlo simulation of net benefits

As an illustration, Figure 2 shows a matrix $\widehat{NB}$ for a small MC simulations with only 5 scenarios (the actual MC simulation used to produce the results in Figure 1 consists of 10^4 scenarios). In scenarios 2 and 5, the legacy application has higher net benefit than the new system and would therefore be prefered by a decision maker with total perfect information.

Information value theory also defines the expected value of *partial* perfect information, *i.e.* perfect information about a subset of the model parameters [53, 20], and the expected value of (partial or total) *imperfect* information, *i.e.* information that reduces uncertainty but without completely eliminating it [36]. In this paper, we will only use the expected value of perfect information either total or about a *single* model parameter. The expected value of imperfect information and of perfect information about sets of parameter 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 to inform the decision:

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

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 [49]. The intuition of this defintion is similar to that of EVTPI. Note that the outer expectation in this definition is taken with respect to the probability distribution of Θ, whereas it was over the probability distribution of Ω for *EVTPI*. The *EVPPI* is always positive or zero.

Computing $EVPPI(\Theta)$ is harder than computing *EVTPI* because it requires an integration (to compute the expectation over $Theta$) of a maximisation over all alternatives of another integration (to compute the expectation in $f(a,\theta)$ over all other model parameters distributions). The classic approach involves a computationally expensive two-level MC simulation (one level for each integration). In this paper, we instead rely on a recent more 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 without the need for additional MC simulation [49]. This algorithm works by first finding 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 the average of these average gains weighted by the proportion of simulation that falls into each segment.

Measuring expected information value is an alternative to sensitivity analysis. There are several variants of sensitivity analysis. 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 [42]. 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 [21].

2.3 The Impact of Information on Risk

Using additional information to maximise expected *NB* has an impact on risk. This impact could be favourable (when selecting an alternative with highest expected *NB* also reduces risk) or unfavourable (when selecting an alternative with highest *NB* increases risk). Measuring this impact give additional information to decision makers about the potential benefits or risks of obtaining additional information.

We have extended information value to include a 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 exposition simple, we'll define the impact of perfect information on risk with respect the a risk measure $Risk(a) = P(\neg F(a,\omega)$ where $F(a,\omega)$ is a predicate that is true when alternative a fails when parameter values are ω. For example, for $LP(a)$, $F(a,\omega)$ is $NB(a,\omega) \leq 0$. 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 selected to be one with the 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* (assuming an NB-optimisation strategy) 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 $\Delta_{TPI}(Risk)$) on each segment of Θ values to compute the Δ in *Risk* in each segment 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 £0.64*m*, 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 arise 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 among a set, they typically involve a multitude of interrelated design decisions concerning choices among alternative architectural styles, design patterns, technologies, and responsibility assignments [55, 57]. This leads to an exponential increase in the number of candidate solutions; for example, if the problem involves 10 design decision 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 [6].

In the objective space, software design decisions typically involve multiple goals that are generally conflicting, hard to defined 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 a priori as complex if not more so than those of software design decisions. 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 [6].

The models relating the design decision options to stakeholders' goal 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 [38] and CBAM [37, 42] 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 [43].

SAS Case Study. We apply our method on a case study of software architecture decisions presented at ICSE 2013 [16].

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 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 — even though uncertainty elicitation experts have criticised it as unreliable [45] — 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 [16].

To facilitate comparison between the approaches, we will apply our method on the same model and data as those used by the GuideArch method [15].

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 [30, 38]. The result is a multi-objective architecture decision model.

Definition. A *multi-objective architecture decision model (MOADM)* 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; we define a *candidate architecture* to be a function $a : D \to \cup_{d \in D} O_d$ that maps each design decision d to a single option in O_d, and we note A the set of all candidate architectures[1];
- C is a set predicates capturing dependency constraints between design decisions such as prerequisite, mutual exclusion, and mutual inclusion relations [61, 50]);
- 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 value ω.

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

Optimisation goals may include software quality attributes such as performance, reliability, *etc.* and stakeholders goals in the application domain 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 [40, 30] are typical examples of goal evaluation functions. These models have parameters, such as the reliability of each components 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 will be assigned probability distributions.

In quantitative goal-oriented requirements models [40, 30], 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 [57].

SAS Case Study. The SAS design team identified the SAS design decisins, their options and optimisation goals shown in Figure 3. Following an approach similar to that used in many goal-oriented decision models [2, 4, 19, 27, 56], they defined the evaluation function for each goal to be 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 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 a inaccurate measure of the true reliability. One 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 measures?).

In order to separate issues concerning the validity of the SAS decision model from discussions concerning the benefits of alternative decision support methods, we will temporarily assume this MOADM to be valid. We will 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 [29]. Since a MOADM could have a large number of optimisation goals, one way to simplify the prob-

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 [16]

lem is to convert the MOADM into a simpler cost-benefit decision model [37, 42]. The cost-benefit model allows software architects to relate design decisions and levels of goal sanctification 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. The cost and benefit functions have to be defined by the software architects in collaboration with project stakeholders. 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 to achieved levels of goal attainments.

A problem with many cost-benefit models is that they a priori exclude from their equations costs and benefits that are perceived to be too hard to quantify and measure. For example, they would omit cost and benefits associated to security, employee productivity, company reputation, *etc.*. To be most useful, cost-benefit models should include the hard-to-measure factors that are important to the decision so that uncertainty about these factors 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 [26, 36, 1].

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 [52]. 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 level of 1 denotes the highest possible level of stakeholders' satisfaction, a preference level of 0 denotes the worst possible level. For example, if g is the response time of a web application, a candidate architecture may receive a preference of 1 if its average response time is less than 1 second and a preference of 0 if its average response time is above 10 seconds. The preference function can then be constructed as a linear or s-shape function between the goal attainments corresponding to the lowest and highest preference [48]. This approach, or a close variant, is found in many requirements engineering methods [2, 4, 19, 27, 56].

An advantage of defining utility as a weighted sums of goal preferences is that it is extremely easy to apply. Its biggest inconvenient, however, is that the utility scores correspond to no physical characteristics in the application domain making them hard to interpret and making the utility function impossible to validate empirically because utility scores cannot be validated against some real measure of utility. In other words, the utility functions are not falsifiable [47]. 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 [6].

When a utility function exist, 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 [6]: $\mathit{benefit}(a,\omega) = K \times U(a, omega)$ This trick 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 SAS design team used 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 assume 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 architectures.

In a cost-benefit model, these risks can be measured as 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) do not exceed some thresholds.

[2]The GuideArch approach assigns to each architecture a a score $s(a)$ to be minimized rather than maximise. To facilitate exposition and relation to other work, we convert the GuideArch score to a utility score to be maximised. We have reproduced the GuidedArch method on the SAS model and verified our change did not affect the results.

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. For every goal, we assume the project client has agreed with the software architects on the minimum level of goal attainment required to avoid failure, noted $must(g)$ (eliciting this value is part of the Planguage, QuPer and KAOS requirements engineering methods [26, 40, 48]). The goal failure risk of a candidate architecture a is then defined as $GoalFailureRisk(g,a) = P(v(g,a,\omega) * must(g))$ where $*$ is $\leq$ or $\geq$ depending on whether g is to be maximised or minimized, respectively.

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:

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

where $must(g)$ is the level of goal attainment below which stakeholders would consider the goal to be failed. 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 [26, 40, 48]).

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 whatever goals are defined in the multi-objective architecture decision model. These goals may include concerns related to software development costs and schedule.

SAS Case Study. The SAS model has no definition of risk and does not specify *must* values for any of its goals. Eliciting such values would be problematic given that the goal lack definitions in terms of observable system characteristics. We thus decided to define the $must(g)$ values relative to the goal level attainment of some baseline architecture whose goal attainments would be equal to those of the existing system. The new system to be developed as to be at least as good as the current system on all goals, otherwise the project would be viewed as failed. We have selected the baseline architecture to be the architecture with the lowest expected net benefit among the top 5% architectures in terms of expected net benefits.

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, there exist simple, reliable methods for performing this elicitation [44].

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 defining triangular fuzzy value functions which are equivalent to triangular probability distributions. A recommended alternative to using triangular distributions is to infer a probability distribution from the 3 point estimates using Bayesian reasoning [44]. For the purpose of comparing our approach with GuideArch, we assume the decision makers' parameters uncertainties are adequately represented by the triangular probability distribution of the SAS model.

3.5 Shortlisting Candidate Architectures

The next step consist 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 should be used. 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 the loss probability and probable loss magnitude. Software architects may select any number of criteria. However, keeping the number of criteria below 3 facilities the generation and visualisation of the shortlist.

Software architects may also specify for each criteria a resolution margin to resist against specious differentiation when comparing alternatives. For example, setting the resolution margin for financial objectives such as expected net benefits and costs to £10,000 means that the shortlisting process will ignore any differences of less then £10,000 when comparing candidate architectures net benefits. These resolutions margins make our shortlisting process robust against statistical errors due to the MC simulations and modelling errors due to simplifications in the model equations.

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 outpefroms a by the resolution margins on *all* criteria at the same. 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 [29] useful 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.

To compute these Pareto-optimal alternatives, we have adapted an classic algorithm for computing Pareto-optimal sets [39] to deal with resolutions margins when comparing alternatives. The advantages of this algorithm over evolutionary algorithms such as NSGA2 [10] more commonly used in search-based software engineering [28] are that it always returns the exact Pareto-optimal set instead of an approximation and that it is easier to implement as it doesn't requires defining and tweaking cross-over and mutation functions. Its inconvenient is that it requires computing the shortlisting criteria values for all alternatives in the design space, whereas evolutionary algorithm explore only a subset of this space.

Our computation of the Pareto-optimal alternatives uses MC simulations to compute the criteria for each alternatives. In practice, our implementation generates the $\widehat{NB}$ for the full design space using MC simulations as described in Section 2, then extract the Pareto-optimal set from this matrix. 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. Our implementation is in `R`, an interpreted programming language for statistical computing. Performance could be greatly improved by porting our MC simulation to `C`.

Once the Parto-optimal shortlist is generated, software architects could further prune this shortlist by restricting their attention to a specific area of the Pareto-optimal strip. For example, they could decide to restrict their attention to alternatives that fall within a specific budget range.

SAS Case Study. Figure 4 shows the Pareto-optimal strip for the SAS candidate architectures evaluated with respect to expected

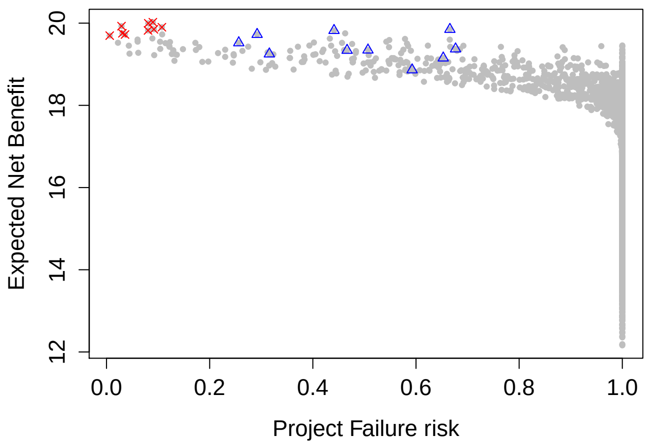

Figure 4: Comparing our shortlisted architectures (red crosses) against GuideArch top 10 (blue triangles). The grey circles denote all other candidate architectures.

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.

net benefit and project failure risk. The resolution margins for each criteria are set at 0.1 and 1%, respectively. The red triangles 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. We observe important differences: our shortlists identifies candidate architecture with slightly higher expected net benefit and much lower project risk than the GuideArch top 10 architectures.

We explain the difference between the two shortlists as follows. Of course, 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 criteria. Instead of evaluating criteria against their expected net benefit (or equivalently their utility score) and some measure of risk, GuideArch ranks candidate architecture according to a single criteria corresponding to an uncertainty-adjusted score defined as the weighted sum of the architecture pessimistic, most 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 (the expected net benefit for an architecture is the mean of all its net benefits in the MC simulation; some of these benefits will be higher, others lower than the expected value). We believe 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 Identifying Open and Closed Design Decisions

Shortlisting a set of candidate architecture may already concludes 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, one can use a method we've developed previously to cluster architectures according to their similarity in terms of design decisions [58].

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

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 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 will come back to this issue below.

SAS Case Study. In the SAS decision models, EVTPI is only 0.05 out of a highest expected net benefit of 20 (*i.e.* 0.25% of the highest expected net benefit). However, EIRTPI is 9%, the same value as the project failure risk for the architecture with highest expected net benefit, which means that the impact of perfect information is to reduce project failure risk to zero. Figure 6 shows all the architecture goal evaluation variables with EVPPI higher than zero. Since these EVPPI are small, the table shows their ratio to EVTPI instead of their absolute value. The ramp up time and battery usage of 4 of the 9 shortlisted architectures are shown to have, in relative,

	$EVPPI/EVTPI$	$\Delta_{PPIP/TPI}(Risk)$
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

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 during decisions, 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 to weights and have therefore replaced their constant value by a triangular distribution of parameters $(0, w(g), w(g))$ where $w(g)$ is the constant goal weight estimated by the SAS design team resulting in a linearly decreasing probability distribution from $w(g)$ to 0. We observed this addition of uncertainty roughly doubled the EVTPI. In our all experiments, the EVTPI remains small. This is mostly due to the little differences in net benefit that exist among the shortlisted architectures for all possible parameter values even when most of the model parameters are increased. If we had confidence in the validity of the model utility function, this result would mean that there would be very little value in reducing uncertainty before deciding among the shortlisted architectures. Decision makers may however have high uncertainty about the validity of the SAS model equations defining the utility function and goal attainments.

Currently, measuring expected information value cannot deal with model uncertainty [12]. For requirements and architecture decision problems there would be high benefits in being able to do so. 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 is a more fine-grained performance model. We have started exploring how to extend our method to deal with model uncertainty in such a way.

4. EVALUATION

We evaluate our method with respect to its correctness, performance and scalability, applicability, and cost-effectiveness. We discuss benefits over related work in the next section.

Is our method correct? One must distinguish correctness of the decision *method* from the correctness of the decision *models* to which the method is applied. Our method produces correct estimations of the candidate architectures expected net benefit, risks, and expected information value *relative to the correctness of the decision models*. If the goal evaluation functions and cost-benefit functions of the decision models are wrong, then our estimations will also be wrong. As mentioned in closing the previous section, we intend to extend 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. This would notably allow us to measure quantitatively when a model is good enough to inform decisions.

Our definitions of expected net benefit, risks, and expected information values are grounded on well-established principles from Bayesian probability theory and decision analysis. Our tool computes these quantities using MC simulation which introduces bounded and measurable simulations errors [41, 46]. 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.

Is our method fast enough and does it scale? We have shown our method is fast enough to analyse a real software design decision problem with around $70,000$ candidate architectures, 8 software design goals, and 175 model parameters. The performance bottleneck of our method is the MC simulation. This did not cause any concern for the SAS case study where the goals evaluation functions are extremely simple but it could become an issue with more complex evaluation functions. Obvious ways of addressing this bottleneck exist: we could use an evolutionary search-based algorithm to reduce the number of candidate architectures to evaluate and parallelize the goal evaluation computations. Another interesting avenue is to reduce the number of scenarios in the MC simulation which would reduce computation time but increase the simulation error. By explicitly including the simulation error in the decision model, we could then compute the expected information value of reducing the simulation error by increasing the simulation size. If this expected value is low, there is no need to increase the simulation size. In this paper, we have focussed on presenting a novel method for requirements and architecture decisions, not yet on optimising its performance. This would require a systematic scalability analysis [14] preferably informed by real case studies rather than academic models or synthetic data. We intend to perform such study in the near future.

Is our method applicable in practice? We have illustrated our method on a real design decision problem from the literature, but we have not yet shown that it is applicable by software architects in actual software engineering projects. We are in the process of identifying and selecting partners and projects to perform such study.

At the moment, we see no critical threats to its applicability. Our method takes as input decision models that correspond to those already produced by other requirements engineering and architecture methods [40, 38, 42, 17]. 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 [44]. Our analysis outputs need to be easily interpretable by decision makers. Although the concepts of risk, Pareto-optimality and information value can be misunderstood, we see no insurmountable obstable here.

Perhaps the most important objection against applicability will come for readers arguing that "Not everything that matters can be measured". If not everything that matters can be measured, then our method, which asks software architects to define measurable goals, is either inapplicable or will lead them to ignore important factors but cannot be measured. Hubbard gives a powerful counterargument to this objection [36]. In summary, he argues the objection is most often caused by a misunderstanding of the nature of measurement: a measurement is a reduction of uncertainty; it is not an absolute elimination of uncertainty. Therefore, any observation that reduces uncertainty even a little is already a measurement. He then argues that everything that matters *can* be measured because (1) if something matters, it must have some direct or indirect observable effects in the world, and (2) if something is observable then it is measurable. The important factors that some consider immeasurable and prefer to deal with qualitatively can in fact be measured and reasoned about with scientifically. Reducing uncertainty about difficult to measure factors, even by a small margin, may in fact be more valuable to a decision than reducing uncertainty about easy to measure factors about which a lot is already known.

Is our method cost-effective? The next evaluation stage is to demonstrate the cost-effectiveness of decision analysis methods in requirements and architecture decisions. A method can be applicable and even applied 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 mean 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 evidences, 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 knowledge uncertainty by relying on point-based estimates of their models parameters [4, 17, 19, 27, 40, 56, 60]. By simply replacing point-based parameter estimation 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 dealing the uncertainty in software architecture decisions, notably CBAM [37, 42] and GuideArch [16]. The first two steps of our method are equivalent to the model elaboration steps in CBAM. Our method differs from CBAM in that it relies on sound reliable techniques for eliciting probability distributions; it includes explicit definition of risks with respect to which alternatives are evaluated; it shortlists candidate architectures based on multiple objectives (*e.g.* ENB and Risk) instead of assuming a single ranking criteria; 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 possible ranges of values. By making their data and decision models fully available, the authors of the GuideArch method have greatly facilitated the development and evaluation of our method. Their SAS model is the only complete model of software design decisions under uncertainty we found in the literature. Our method differs from GuideArch in that it does not presuppose a fixed structure for the decision models, it encourages the use of measurable decision criteria and falsifiable decision models, its defines and measures decision risks, and it computes expected information value.

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.* [31, 25, 40, 30]). Philosophers and statisticians use the terms epistemic and aleatory uncertainty, respectively [45]. 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 for 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 [18] and uncertainty in requirements definitions for adaptive systems [59]. These are different concerns and meanings of uncertainty than those studied in this paper.

Graphical decision-theoretic models [13] and Bayesian networks [24] provide general tools for decision making under uncertainty. They have been used to support software decisions regarding development resources, costs, and safety risks [22, 23] but not to support requirements and architecture decisions. We have not used these tools to support our method because they deal with discrete variables only; they would have require transforming our continuous variables such as cost, benefit and the levels of goal attainments into discrete variables.

Boehm's seminal book on software engineering economics devotes a chapter to statistical decision theory and the value of information [7]. 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 [3, 8, 22, 51] 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 believe 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 argue that statistical decision analysis and information value theory provide the right set of tools to manage uncertainty in complex software engineering decisions, at the requirements and architecture levels but possibly also in other areas, such as testing, where critical decisions must be made by analysing risks arising out of incomplete knowledge. We are currently exploring how to extend our method to deal with model uncertainty in addition to parameter uncertainty.

7. REFERENCES

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

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

[3] A. J. Albrecht. Measuring application development productivity. In *Proceedings of the Joint SHARE/GUIDE/IBM Application Development Symposium*, volume 10, pages 83–92. SHARE Inc. and GUIDE International Corp. Monterey, CA, 1979.

[4] 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.

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

[6] G. Baio. *Bayesian Methods in Health Economics*, volume 53. CRC Press, 2012.

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

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

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

[10] 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.

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

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

[13] 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.

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

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

[16] 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.

[17] 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.

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

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

[20] J. C. Felli and G. B. Hazen. Sensitivity analysis and the expected value of perfect information. *Medical Decision Making*, 18(1):95–109, 1998.

[21] J. C. Felli and G. B. Hazen. Sensitivity analysis and the expected value of perfect information. *Medical Decision Making*, 18(1):95–109, 1998.

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

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

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

[25] C. Ghezzi, L. S. Pinto, P. Spoletini, and G. Tamburrelli. Managing non-functional uncertainty via model-driven adaptivity. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 33–42. IEEE Press, 2013.

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

[27] P. Giorgini, J. Mylopoulos, E. Nicchiarelli, and R. Sebastiani. Reasoning with goal models. In *Conceptual ModelingâĂŤER 2002*, pages 167–181. Springer, 2003.

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

[29] 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.

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

[31] 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.

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

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

[34] D. Hubbard. The it measurement inversion. *CIO Enterprise Magazine*, 1999.

[35] D. Hubbard. *The Failure of Risk Management: Why It's Broken and How to Fix It*. Wiley, 2009.

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

[37] R. Kazman, J. Asundi, and M. Klein. Quantifying the costs and benefits of architectural decisions. In *Proceedings of the 23rd international conference on Software engineering*, pages 297–306. IEEE Computer Society, 2001.

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

[39] 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.

[40] 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.

[41] 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.

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

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

[44] 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.

[45] A. O'Hagan and J. E. Oakley. Probability is perfect, but we can't elicit it perfectly. *Reliability Engineering & System Safety*, 85(1âĂŞ3):239 – 248, 2004. <ce:title>Alternative Representations of Epistemic Uncertainty</ce:title>.

[46] 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.

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

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

[49] 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.

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

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

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

[53] M. Strong and J. E. Oakley. An efficient method for computing single-parameter partial expected value of perfect information. *Medical Decision Making*, 33(6):755–766, 2013.

[54] 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.

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

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

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

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

[59] J. Whittle, P. Sawyer, N. Bencomo, B. H. Cheng, and J.-M. Bruel. Relax: Incorporating uncertainty into the specification of self-adaptive systems. In *Requirements Engineering Conference, 2009. RE'09. 17th IEEE International*, pages 79–88. IEEE, 2009.

[60] 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.

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