2017 IEEE/ACM 39th International Conference on Software Engineering

RADAR: A Lightweight Tool for Requirements and Architecture Decision Analysis

Saheed A. Busari, Emmanuel Letier
Department of Computer Science
University College London
London, United Kingdom
{saheed.busari.13, e.letier}@ucl.ac.uk

***Abstract*—Uncertainty and conflicting stakeholders' objectives make many requirements and architecture decisions particularly hard. Quantitative probabilistic models allow software architects to analyse such decisions using stochastic simulation and multi-objective optimisation, but the difficulty of elaborating the models is an obstacle to the wider adoption of such techniques. To reduce this obstacle, this paper presents a novel modelling language and analysis tool, called RADAR, intended to facilitate requirements and architecture decision analysis. The language has relations to quantitative AND/OR goal models used in requirements engineering and to feature models used in software product lines. However, it simplifies such models to a minimum set of language constructs essential for decision analysis. The paper presents RADAR's modelling language, automated support for decision analysis, and evaluates its application to four real-world examples.**

***Keywords*-Decision Analysis, Requirements Engineering, Software Architecture, Goal Modelling, Monte-Carlo Simulation, Multi-Objective Optimisation, Search-Based Software Engineering, Expected Value of Information**

I. Introduction

Designing software systems involves deciding what software functions should be provided, what levels of quality requirements should be met, and what software architecture to use to satisfy these functional and quality requirements. Such requirements and architecture decisions often have critical impacts on the software development costs, duration, and the system's ability to satisfy stakeholders' goals [1], [2].

Many software requirements and architecture decisions have to deal with multiple conflicting objectives and are confronted with high levels of uncertainty about the possible impacts of decision choices [3]. Relying on intuitions alone for such critical and complex decisions is not ideal. Intuitive decisions, even by experts, are subject to many cognitive biases and errors [4]. Some of these biases have been shown to happen in software engineering contexts [5].

Using quantitative decision models is therefore a promising approach to improve software requirements and architectural decisions making. Such models help to clarify the decision problem and allow software architects to apply stochastic simulation and multi-objective optimisation techniques to analyse the decisions [3], [6].

One of the biggest obstacles to such quantitative methods, however, is the "modelling problem"; the difficulty of elaborating a quantitative decision model that adequately defines the stakeholders' goals and correctly predicts the impacts of decisions on these goals.

Many requirements and architecture decision methods avoid this problem by relying on generic decision objectives and predefined model equations. For example, the EVOLVE release planning method [7], the CBAM architecture decision method [8], [9], and many search-based methods for requirements selection [10] rely on predefined equations (generally weighted sums) to assign 'cost' and 'value' scores to alternative designs. The cost and value scores in these methods usually represent abstract (non verifiable) quantities rather than financial metrics expressed in monetary units (e.g. in Dollars or Euros). Using generic decision models simplifies the application of these methods, but the generic models generally fail to capture the stakeholders' real objectives and be valid models of the impacts of decisions on these objectives [11]. If the decision model does not reflect the decision-maker's objectives, it is unlikely to provide useful guidance. We therefore need decision methods that support the elaboration and analysis of problem-specific decision models.

Goal-oriented requirements engineering provides systematic methods to guide the elaboration of problem-specific requirements and architecture decision models [12]. The KAOS goal modelling framework allows one to define stakeholders' goals and their relation to software requirements and domain assumptions in precise terms [1], [13]. Quantitative goal models extend KAOS goal models by describing quantitatively the impacts of alternative requirements and architecture designs on measurable stakeholders' goals [14]. Stochastic simulation and multi-objective optimisation can then be used to analyse decisions in quantitative goal models [6]. Currently, however, such analysis requires the engineer to manually develop the model simulation in a general programming language. Previous work used MATLAB and R [6], [14]. The complexity of the KAOS goal modelling framework and its quantitative extension may also hinder their use in practice.

Our objective is to develop a lightweight decision modelling language and automated analysis tool for requirements and architecture decision analysis. The result is RADAR, the Requirements and Architecture Decision AnalyseR. RADAR's modelling language is a simplified form of quantitative goal models designed to be similar to simple equations that software architects use for back-of-the-envelope calculations (i.e.

1558-1225/17 $31.00 © 2017 IEEE
DOI 10.1109/ICSE.2017.57

rough, imprecise calculations) [2]. RADAR, however, provides sophisticated analysis that cannot be performed through back-of-the-envelope calculations: it allows analysing uncertainty through Monte-Carlo simulations, shortlisting Pareto-optimal solutions through multi-objective optimisation, and computing expected value of information that can be used to decide whether to seek more information or perform a more detailed analysis before making a decision [3].

RADAR is designed to support the requirements and architecture decision method introduced in previous work [3]. In previous work, applying this method requires modellers to develop a decision model and its simulation in the R programming language. This impairs model readability and forces modellers to consider implementation issues instead of focussing solely on the conceptual decision problem. RADAR's novelty with respect to that method are: (i) its modelling language that frees software engineers from having to implement their decision models in R or another general programming language (Section IV-A); (ii) the automated generation of AND/OR refinement graphs and decision graphs from the model's equations (Sections IV-B and IV-D); (ii) the automated inference of decision dependencies (Section IV-D); (iv) the minimisation of the optimisation problem's search space by automatically inferring the set of minimal and complete solutions associated to a RADAR model (Section IV-C); and (v) the implementation of techniques for simulating and optimising the model's decisions and for computing the expected value of total and partial perfect information (Section IV-E). The decision method itself has not changed.

II. A Motivating Example

To motivate our approach and illustrate our modelling language and tool, we will use the example of the design of a plastic card fraud detection system. This example is based on our previous experience of analysing the scalability of a commercial financial fraud detection system [15], [16].

Plastic card fraud detection systems are used by banks to detect when plastic card accounts may have been compromised by fraudsters who are using the account to steal funds. Design decisions for such systems include:

- the *processing type* that can be continuous or batch; continuous processing analyses transactions individually as they arrive, whereas batch processing performs an overnight analysis of the transactions that occurred during the day;
- the *fraud detection method* which can be a two-class supervised classification method in which a classifier is trained from samples of past fraudulent and non-fraudulent transactions, or a non-statistical rule-based method that flags transactions matching configurations known to be high risk;
- if the classifier fraud detection method is chosen, the *alert threshold* defines some threshold above which the classifer should flag a transaction as suspect. A low alert threshold means more alerts will be generated and thus a higher ratio of false alerts.
- the *blocking policy* that can include blocking an account as soon as the fraud detection method flags a transaction as suspected fraud, or only blocking the account after the suspected fraud has been confirmed by human investigators.

These decisions impact the number of alerts generated and the speed at which compromised accounts are blocked, which ultimately affects how much effort the bank needs to devote to manually investigate alerts and how much money it loses to fraud. The optimisation of plastic card fraud detection systems typically include two conflicting concerns: minimising financial loss, and minimising manual investigation costs [17].

Deciding what combination of processing type, fraud detection method, alert threshold and blocking policy to use is not trivial. The problem is complicated by uncertainty about domain quantities, such as the ratio of compromised accounts, uncertainty about the impact of decisions on future financial loss and investigation costs, and the conflicting nature of these two concerns.

III. RADAR: An Overview

Before defining RADAR in detail, which we do in Section IV, we first illustrate the application of RADAR on a complete, simple, but illustrative example. The purposes are to give a high-level overview of the language and to provide background on our decision analysis method [3].

A. Developing a decision model

Imagine having to perform a cost-benefit analysis for deciding whether or not to refactor the architecture of an existing application. With the current architecture, the application generates relatively predictable benefits. Refactoring creates the possibility of generating much higher benefits, but the refactoring costs and benefits are highly uncertain.

A RADAR model for this decision problem might look like this:

$$
\begin{aligned}
&\textbf{Objective}\ \ \textbf{Max}\ \mathrm{ENB} = \mathbf{EV}(\mathrm{NB});\\
&\textbf{Objective}\ \ \textbf{Min}\ \mathrm{LP} = \mathbf{Pr}(\mathrm{NB} < 0);\\
&\mathrm{NB} = \mathrm{Benefit} - \mathrm{Cost};\\
&\mathrm{Cost} = \mathbf{decision}(\text{“Architecture choice”})\{\\
&\quad \text{“As-is”} : \mathbf{deterministic}(0);\\
&\quad \text{“Refactoring”} : \mathbf{normalCI}(1,\ 5);\\
&\}\\
&\mathrm{Benefit} = \mathbf{decision}(\text{“Architecture choice”})\{\\
&\quad \text{“As-is”} : \mathbf{normalCI}(0.9,\ 1.1);\\
&\quad \text{“Refactoring”} : \mathbf{normalCI}(1,\ 9);\\
&\}
\end{aligned}
$$

The language keywords are in bold. The first two lines define the optimisation objectives: maximising expected net benefit (ENB) and minimising loss probability (LP). The function $\mathbf{EV}$ denotes the expected value (or mean) of a random variable and $\mathbf{Pr}$ denotes the probability of a Boolean expression. The model's third line defines net benefit (NB) as the difference between benefit and cost.

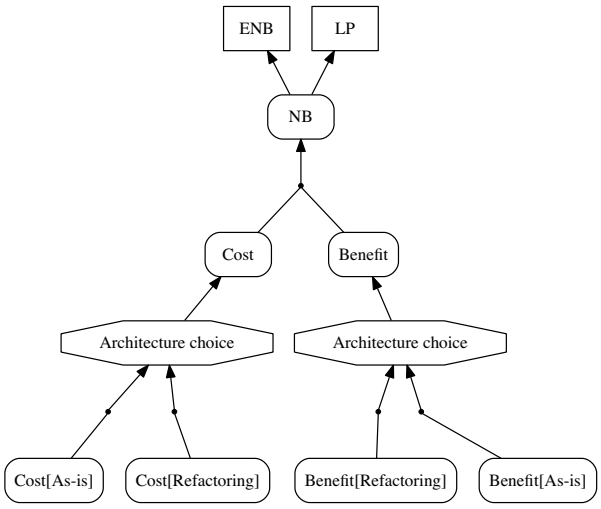

Fig. 1. AND/OR refinement graph for the cost-benefit analysis example

The next four lines state that cost depends on the architectural choice. If the choice is to keep the as-is architecture, we assume the cost to be zero. The **deterministic** keyword means we believe this cost to be certain. If the choice is to refactor, we believe the cost has a 90% chance of being between £1m and £5m. The expression $\mathbf{normalCI}(1,\ 5)$ means the cost follows a normal distribution whose 90% confidence interval is between 1 and 5. Similarly, the last four lines state that benefit depends on the architecture choice and records our beliefs about the benefit's likelihood for the as-is and refactored architecture.

Probabilities in our approach are Bayesian; probability distributions denote the decision makers' beliefs about the likelihood of uncertain quantities and events. These beliefs can be informed by subjective judgements, objective data, or a combination of both. Bayesian methods typically start with probability distributions informed by subjective judgements alone, then update the distributions (using Bayes rule) as new data and information becomes available [18], [19].

Reliable methods exist for eliciting a person's beliefs about uncertain quantities or events, and modelling these beliefs as probability distributions [20]. A recommended simple approach consists in eliciting 90% confidence interval as used above [21]. For these elicitation methods to be reliable, people providing estimations have to be 'calibrated' on a set of estimation exercises intended to mitigate their under- or over-confidence biases.

To help visualising the model structure, RADAR automatically generates the AND/OR refinement graph and decision graph of Fig. 1 and 2. RADAR AND/OR refinement graphs are equivalent to quality variables AND/OR refinement graphs in quantitative goal models [14]. In RADAR, rectangles denote objectives, rounded rectangles denote random variables (i.e. variables characterized by a probability distribution rather than a single value), a black dot denotes an AND-refinement, and an

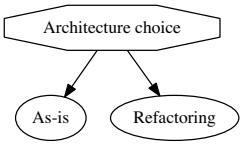

Fig. 2. Decisions dependency graph for the cost-benefit analysis example

octagon denotes an OR-refinement. An arrow from a variable to an objective denotes that the objective refers to that variable. The leaf nodes in the AND/OR refinement graphs are the model parameters. Their values are defined by probability distributions. In our example, Fig. 1 shows that the objectives ENB and LP both refer to NB, that NB depends on Benefit *and* Cost (an AND-refinement), while Benefit depends on Benefit[As-is] *or* Benefit[Refactoring] based on which option is chosen (an OR-refinement).

RADAR decision graphs play a similar role to feature models in software product lines [22]; they help us visualise the model decisions, their options and possible decision dependencies. In our refactoring model, the decision graph is extremely simple because the model includes a single decision and no decision dependency. Fig. 6 shows a more interesting decision graph for our fraud detection example. Octagons denote decisions; ovals denote options; the arrows from a decision to options denote all options available for that decision; an arrow from an option to a decision denotes that the decision has to be made only for system designs where that option has been selected. The options in a decision are mutually exclusive.

B. Analysing decision models

RADAR supports a decision analysis method that consists in first shortlisting a set of Pareto-optimal solutions through simulation and multi-objective optimisation, then computing the expected value of information to evaluate whether to seek additional information before making a decision between the shortlisted candidates [3]. This section provides a brief overview of this method and how RADAR supports it.

Fig. 3 shows RADAR's analysis results for our small refactoring model. The first part shows the results of RADAR's optimisation analysis. It lists the optimisation objectives and the objective values for the two architecture choices: refactoring has an expected net benefit of £2m, but a loss probability of 23%, whereas keeping the current architecture has an expected net benefit of £1m but the loss probability is zero.

In this small example, we have only two solutions to choose from. Larger problems such as the fraud detection problem of Section II have a larger number of solutions. Before displaying the optimisation analysis results, RADAR shortlists the set of Pareto-optimal solutions and presents only those to the decision makers. A solution is Pareto-optimal if there is no other solution that is better on all objectives simultaneously [3]. In our small example, a solution is thus Pareto-optimal if no other solution has both higher expected net benefit and lower loss probability. Here, both solutions are

Optimisation Analysis

Objective: **Max** ENB
Objective: **Min** LP

Architecture choice	ENB	LP
Refactoring	2	0.23
As-is	1	0

Information Value Analysis

Objective: **Max** ENB
EVTPI: 0.64

Parameter	EVPPI
Benefit[Refactoring]	0.54
Cost[Refactoring]	0.14
Benefit[As-is]	0
Cost[As-is]	0

Fig. 3. Analysis results for the cost-benefit analysis model (ENB = Expected Net Benefit; LP = Loss Probability; EVTPI = Expected Value of Total Perfect Information; EVPPI = Expected Value of Partial Perfect Information).

Pareto-optimal because none of them is better than the other on both objectives.

The second part of Fig. 3 shows the result of information value analysis [3], [23]. In many decision situations, we may be able to collect and analyse additional data to reduce our uncertainty before making a decision. Additional data collection and analysis, however, are worthwhile only if their cost is lower than the value of the new information they bring.

Information value analysis gives upper bounds on the value of additional data collection and analysis to our decision problem. The *expected value of total perfect information* (EVTPI) is a theoretical measure of the expected gain in some objective value (usually, maximising net benefit) that would result from having access to perfect information about all model parameters, that is from having access to an oracle who could tell us the exact values of all model parameters. The EVTPI gives an upper bound to the information that would result from additional data collection and analysis. Similarly, the *expected value of partial perfect information* (EVPPI) is the expected gain in some objective value resulting from having access to perfect information about a single model parameter. It gives an upper bound to how much we should spend to reduce uncertainty about that model parameter. We refer readers to previous publications for more formal and detailed explanation of these concepts [3], [23], [24], [25].

Analysing the expected value of information is important because it helps mitigate a measurement bias, known as measurement inversion, where decision makers would spend sometimes considerable efforts measuring quantities with low or even zero information values but disregard measuring quantities with high information value [21]. This bias has notably been observed in a study of 20 IT project business cases [26]. This study cites the effort spent by an organisation conducting detailed measurement of software development productivity as an example of measurement with very low information value, whereas quantities with high information value that are not measured at all are typically those related to benefits that are wrongly perceived to be intangible.

In our example, we evaluate information value with respect to maximising expected net benefit. The EVTPI is £0.64 million. Spending a small fraction of that amount on reducing uncertainty could have high value. The EVPPI values show that reducing uncertainty about the benefits of refactoring has by far the highest value (£0.54m). By contrast, reducing uncertainty about refactoring cost has little value and reducing uncertainty about the current architecture has no value.

One way to reduce uncertainty about the benefits of refactoring would be to elaborate a finer-grained decision model by refining the Benefit variable into lower-level variables (e.g. customers retention and acquisition rates, savings in software maintenance costs) and potentially identifying finer-grained architecture decisions corresponding to alternative ways to refactor the existing architecture. This would trigger a new decision analysis. The cycle of model refinement and analysis would eventually stop when the remaining expected value of perfect information is too low to justify further analysis.

IV. RADAR Language and Analysis

Now that we have a general overview, we introduce the modelling language and automated analysis in more detail.

A. The Modelling Language

A RADAR model is composed of a set of objective definitions and variable definitions.

1) Objective Definition: An *objective definition* has the form

$$\textbf{Objective}\ \ (\textbf{Min} \mid \textbf{Max})\ \mathrm{Name} = \mathrm{Statistic(X)}$$

where Name is the objective name, **Min** or **Max** declares whether the objective function should be minimised or maximised, and Statistic(X) is a statistical measure on a single random variable X. Statistical measures include:

- $\mathbf{EV}(\mathrm{X})$ denoting the expected value of X;
- $\mathbf{Pr}(\mathrm{X} \sim \mathrm{x})$ denoting the probability that $\mathrm{X} \sim \mathrm{x}$ where $\sim$ is $\leq$, $<$, $=$, $>$, or $\geq$;
- $\mathbf{percentile}(\mathrm{X}, \mathrm{i})$ denoting the i^{th} percentile of X, i.e. the value x such that $\mathbf{Pr}(X \leq x) = i$.

We saw examples of the use of first two types of statistics in the refactoring model of Section III. Percentiles are useful statistics for measuring risk. For example, in the refactoring example, the Value at Risk (VaR) can be defined as:

$$\textbf{Objective}\ \ \textbf{Min}\ \mathrm{VaR} = \mathbf{percentile}(\mathrm{NB}, 5)$$

In our example, the VaR of refactoring is -2.6, i.e. the chance of losing more than £2.6m is less than 5%.

2) Variable definition: A *variable definition* is either an AND-refinement, an OR-refinement, or a parameter estimation.

An *And-refinement* has the form

$$\mathrm{X} = F(X_1, ..., X_n)$$

where X is a variable and $F(X_1, ..., X_n)$ is an arithmetic or Boolean expression involving variables $X_1, ..., X_n$. The equation defining NB in our cost-benefit analysis is an example of And-Refinement where NB is defined as $\mathrm{Benefit} - \mathrm{Cost}$.

An *OR-refinement* has the form

$$\begin{aligned}&\mathrm{X} = \mathbf{decision}(label)\{\\ &\quad Option_1 : Expression_1;\\ &\quad ...\\ &\quad Option_n\text{: } Expression_n;\\ &\}\end{aligned}$$

where $label$ is the decision name, $Option_i$ are option names, and $Expression_i$ is an AND-refinement or parameter estimation defining the value of X if $Option_i$ is selected.

The definitions of Cost and Benefit in the refactoring model are examples of OR-refinement.

Consider also Fig. 4 that shows all OR-refinements in the fraud detection model. The first OR-refinement states that NbrFraudPerAccountBeforeBlocked depends on the blocking policy; the second that NbrFraudBeforeDetection depends on the processing type; etc.

Multiple OR-refinements can refer to the same decision. For example, in Fig 4, both variables ContinuousTrueAlertRate and BatchTrueAlertRate depend on the fraud detection method.

A *parameter estimation* has the form:

$$\mathrm{X} = ProbabilityDistribution$$

where *ProbabilityDisribution* defines a probability distribution for the variable X. Probability distributions include:

- **uniform**(min, max) denoting the uniform distribution between values min and max;
- **triangular**(min, mode, max) denoting the triangular distribution with lower limit min, upper limit max and mode mode;
- **normalCI**(lower, upper) denoting a normal distribution characterised by the lower and upper bounds of its 90% confidence interval (i.e. lower is the 5^{th} percentile and upper the 95^{th} percentile);
- **deterministic**(x) denoting that the variable has the certain value x.

For example, the number of accounts in our fraud detection model is defined by the following parameter estimation:

$$\mathrm{NbrAccounts} = \mathbf{normalCI}(0.9 \times 10^6, 1.1 \times 10^6);$$

A parameter estimation can be used in an OR-refinement to define the value of a variable X when $Option_i$ is selected, thereby introducing a parameter $\mathrm{X}[Option_i]$. For example, in the cost-benefit analysis model, the OR-refinement for Cost introduces the parameters Cost[As-is] and Cost[Refactoring].

B. AND/OR Refinement Graph

The equations in a RADAR model create an AND/OR refinement graph between variables. An AND-refinement relates a variable to the set of variables involved in its definition. An OR-refinement relates a variable to the set of AND-refinement or parameter estimations involved in the OR-refinement definition. As an example, Fig. 5 shows the AND/OR refinement graphs for the fraud detection model fragments in Fig. 4.

The AND/OR refinement graph of a model must be acyclic. The tool generates an error if it detects a circular dependency.

By showing the variable dependencies, the AND/OR refinement graph helps modellers to review and validate the model structure with other stakeholders. Such AND/OR graphs are commonly used in goal-oriented requirements engineering to communicate and validate traceability links between technical software characteristics (e.g. the classifier's true alert rate) and high-level stakeholders' concerns (e.g. financial loss due to fraud) [1].

C. The Design Space

A model's OR-refinement equations introduce a set of decisions and options. Selecting an option for a particular decision replaces all OR-refinements that refer to this decision by AND-refinements or parameters estimation corresponding to the selected option. For example, selecting the "block first" option for the "decision policy" option in Fig. 4 replaces the OR-refinement for NbrFraudPerAccountBeforeBlocked by the AND-refinement corresponding to the "block first" option, i.e. NbrFraudPerAccountBeforeBlocked = NbrFraudBeforeDetection. This idea provides the basis for how we define the set of valid solutions for a RADAR decision model.

Definition (Solution Space). Let D be the set of all decisions declared in the model, $O(d)$ be the set of options associated with decision $d \in D$, and $O = \cup_{d \in D} O(d)$ be the set of all options. We define a *solution* s to be a mapping from decisions to options $s : D \to O$ such that $s(d) \in O(d)$. A solution is *total* if it contains an option for every decision in D (i.e. it is a total function); otherwise the solution is said to be *partial*. We define the *solution space* to be the set of all total solutions.

For example, $s_1 = \{$(blocking policy, block first), (processing type, continuous), (fraud detection method, classifier), (alert threshold, high)$\}$ is a total solution for the fraud detection model, whereas $s_2 = \{$(blocking policy, block first), (processing type, continuous), (fraud detection method, rule-based)$\}$ is a partial solution because it contains no alert threshold decision. For any model, the size of the solution space is $\prod_{d \in D} |O(d)|$. For example, the size of the solution space for the fraud detection model is $2 \times 2 \times 2 \times 3 = 24$.

Applying a solution s to a RADAR model replaces OR-refinements with AND-refinements corresponding to the selected options for all decisions defined in s.

Definition (Design Space). A solution s is *complete* if applying the solution results in a model that has no OR-refinement. A solution s is *minimal* if no subset of s is complete. The *design space* of a RADAR model is the set of all its minimal and complete solutions.

```
NbrFraudPerAccountBeforeBlocked = decision("blocking policy"){
        "block first" : NbrFraudBeforeDetection;
        "investigate first" : NbrFraudBeforeDetection + NbrFraudDuringInvestigation;
}
NbrFraudBeforeDetection = decision("processing type"){
        "continuous" : 1 / ContinuousTrueAlertRate;
        "batch" : NbrFraudsPerCompromisedAccountPerDay / BatchTrueAlertRate ;
}
ContinuousTrueAlertRate = decision("fraud detection method"){
        "classifier" : ContinuousAlertThreshold;
        "rule-based" : deterministic(0,75)
}
BatchTrueAlertRate = decision("fraud detection method"){
        "classifier" : BatchAlertThreshold;
        "rule-based" : deterministic(0,80);
}
ContinuousAlertThreshold = decision("alert threshold"){
        "high" : deterministic(0.9);
        "medium" : deterministic(0,8);
        "low" : deterministic(0,7);
}
BatchAlertThreshold = decision("alert threshold"){
        "high" : deterministic(0.95);
        "medium" : deterministic(0,85);
        "low" : deterministic(0,75);
}
```

Fig. 4. Fragment of RADAR model showing all OR-refinements in the financial fraud detection system

A total solution is always complete, but not all complete solutions are total. For example, s_2 is partial and complete; it is complete because selecting the *"rule-based"* option as a fraud detection method replaces the OR-refinement by an AND-refinement whose subgraph contains no further OR-refinement (our model does not include an alert threshold decision for the rule-based fraud detection method). Obviously, not all partial solutions are complete. For example, the partial solution $s_3 = \{(\text{blocking policy, block first})\}$ is not complete.

Not all complete solutions are minimal. For example, the solution $s_4 = \{(\text{blocking policy, block first}), (\text{processing type, continuous}), (\text{fraud detection method, rule-based}), (\text{alert threshold, high})\}$ is not minimal because its subset s_2 is complete.

The *design space* of a RADAR model defines the set of all valid solutions to be considered during optimisation. Our tool generates the design space of a model by recursion over the acyclic AND/OR refinement graph by merging the set of solutions associated to a variable subgraph. For any model, the size of the design space is always smaller or equal to the size of the solution space. For the fraud detection model, the design space contains 16 solutions.

Note that our decision-based solution encoding is different from the alternative option-based encoding commonly used in search-based software engineering, notably for the problem of selecting optimal designs in software product lines [27], [28], [29]. In the option-based encoding, solutions are encoded as a mapping $s : O \rightarrow \textit{Boolean}$ such that for each option $o \in O$, $s(o)$ denotes whether o is selected or not. Additional constraints must then be added to remove invalid solutions such as those that select two mutually exclusive options. With an option-based encoding, the solution space would include $2^{|O|}$ solutions against $\prod_{d \in D} |O(d)|$ for our decision-based encoding. In the fraud detection model, an option-based encoding would have resulted in $2^9 = 512$ total solutions instead of 24. For a slightly larger model including 10 decisions with 3 options each, an option-based encoding would include $2^{3*10} \approx 10^9$ total solutions, whereas the decision-based encoding would have only $3^{10} \approx 59,000$ solutions, i.e. 0.005% of the number of solutions in the option-based encoding. The benefits of a decision-based encoding over an option-based encoding are thus enormous: the solution space is much smaller and it does not need additional constraints to remove invalid solutions.

D. The Decision Graph

The equations in a RADAR model may create dependencies between decisions. For example, in the fraud detection model, the "alert threshold" decision is dependent on the selection

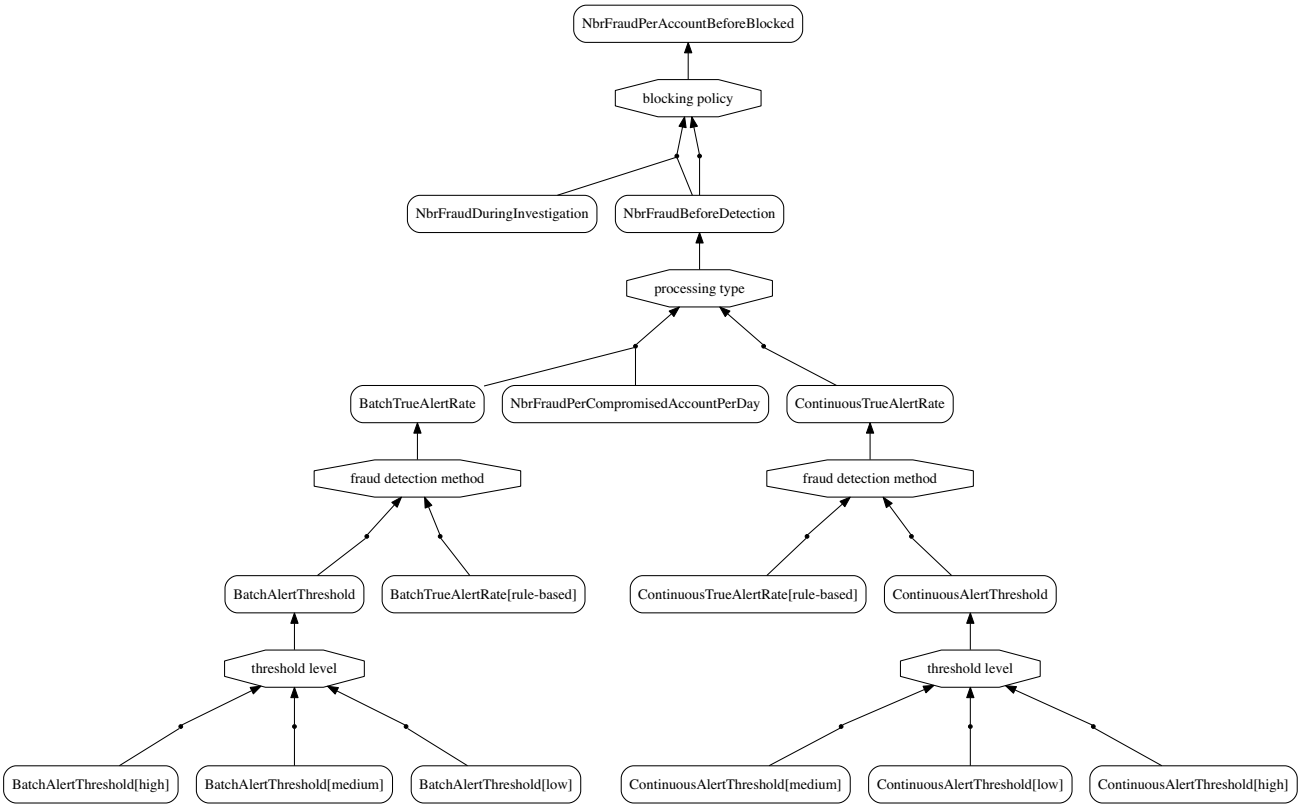

Fig. 5. AND/OR Refinement Graph for the Financial Fraud Detection System

of the "classifier" option in the "fraud detection method" decision.

Definition (Decision Dependency). A decision d_1 is *dependent* on the selection of option x in decision d_0 if, and only if, for all solutions s in the design space, if d_1 is defined then the selected option for decision d_0 is x; formally: $\forall s \in \mathit{DesignSpace} \mid d_1 \in \mathit{dom}(s) \Rightarrow s(d_0) = x$, where $\mathit{dom}(s)$ denotes the domain of the function s, i.e. the set of decisions that have a mapping in s.

Our tool infers decision dependencies by first generating the design space, then checking for dependency between every pair of decisions. To visualise such dependencies, the tool generates a decision diagram showing all decisions, their options, and dependencies between decisions and options. The decision diagram for the fraud detection model is shown in Fig. 6. These diagrams play a similar role to that of feature diagrams in software product lines [22]: they help us visualise a potentially large design space in terms of a smaller set of decisions and options.

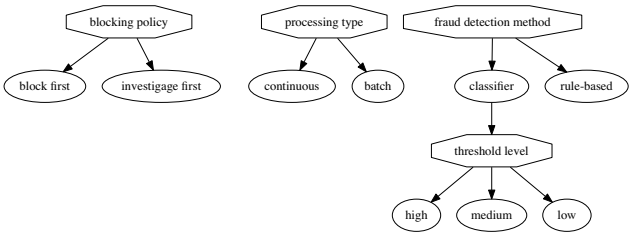

Fig. 6. Decision Graph for the Financial Fraud Detection System

E. Analysing the Model

As illustrated in Section III, the analysis of a RADAR model consists in shortlising its Pareto-optimal solutions and evaluating the expected value of total and partial perfect information.

Performing this analysis involves three components:

1) *The Simulator*: The Simulator provides two related functions: (i) given a candidate solution s, it returns the values of all objectives for s; (ii) given a set S of candidate solutions and a variable X, it returns a simulation vector $\overline{X}$ that contains simulations of X and a simulation matrix $\overline{\overline{P}}$ that contains the simulations for all model parameters used to compute $\overline{X}$. The first function is used by the Optimiser, the second by the Information Value Analyser. Internally, both functions use the same Monte-Carlo simulation where simulations of variables are generated by recursion through the AND/OR refinement equations by selecting the appropriate decision options through OR-refinements. The number N of simulations is set by the user and has a default value of 10^4. Our implementation ensures that, in a given simulation run, all

solutions are evaluated using the same parameter simulation data, which is needed to ensure correctness.

2) *The Optimiser*: given a RADAR model, the Optimiser returns the set of Pareto-optimal solutions together with their objective values. Our implementation identifies the exact set of Pareto-optimal solutions through an exhaustive search of the design space.

As will be shown in Section V, such exhaustive analysis is efficient on all problems we have analysed so far. If future problems cannot be solved through exhaustive search, our architecture supports replacing the exhaustive search strategy by genetic multi-objective optimisation algorithms, such as such as NSGA2 [30], that can deal with larger design spaces but do not guarantee to return the exact set of Pareto-optimal solutions.

3) *The Information Value Analyser*: given an objective to be considered for information value analysis (typically maximising net benefit) and a subset of solutions from the design space (typically the set of Pareto-optimal solutions), the Information Value Analyser returns the EVTPI and the EVPPI of all non-deterministic model parameters with respect to the given objective and set of solutions. Our implementation relies on existing algorithms. EVTPI is computed from the matrix $\overline{\overline{NB}}$ that contains simulations of the objective variable for each solution in the given subset ($\overline{\overline{NB}}[i, j]$ is the value of *NB* in simulation i for solution j). EVPPI of a model parameter x is computed using a recent efficient algorithm that relies only on $\overline{\overline{NB}}$ and the vector $\overline{x}$ containing the simulations of parameter x [25].

For scalability reasons, we perform information value analysis by considering shortlisted solutions only, rather than the whole design space. Considering the whole design space would involve generating a matrix $\overline{\overline{NB}}$ of size $N \times M$ where N is the number of simulations, typically 10^4, and M is the size of the design space which could be up to 59000 for an average problems with 10 decisions having 3 options each. Matrices of such size become difficult to manipulate in memory. Computing expected information value for shortlisted solutions is more efficient (the number of rows M of the matrix $\overline{\overline{NB}}$ is the number of shortlisted solutions) but may generate slightly lower EVTPI and EVPPI values than by considering the whole design space. Our approach amounts to deciding a priori that non-shortlisted solutions will not be considered for further analysis.

Verifiability. RADAR can be downloaded from the tool's webpage (https://ucl-badass.github.io/radar/). The tool is implemented in Java. It uses ANTLR to generate the model parser [31] and produces diagrams in the DOT format that can be visualised and converted to other formats using Graphviz [32]. All models discussed in the paper are available from the tool's webpage. Readers can review the models, replicate their analysis, and perform additional analysis by modifying the models or creating their own.

V. Evaluation

Empirically evaluating a novel modelling language and decision analysis techniques is notoriously difficult. Our evaluation focuses on showing that the RADAR modelling language and analysis techniques are *applicable* to real-world problems. Our discussion about their *usefulness* is, at this stage, speculative.

A. Applicability

To show that our language and tool are applicable to real-world problems, we have applied them to four representative real-world examples: the design of a financial fraud detection system introduced earlier [15], [16], [17]; decisions about system security policies [33], [34], the design of a system to coordinate the deployment of emergency response teams [3], [35], [36], and the design a system to collect and process satellite images [8], [9], [37]. The last two problems have been used previously in the software engineering community to illustrate architecture decision methods [3], [8], [9], [36].

Table I shows the problem size and analysis run-time for each model analysed with $N = 10^4$ simulations. Small models such as the fraud detection and security policy models are analysed in less than a second; a larger model such as the emergency response model is analysed in less than 2 minutes.

By successfully applying our modelling language and tool to these problems, we have shown that:

- **Claim 1:** The RADAR modelling language is expressive enough to model real-world requirements and architecture decision problems;
- **Claim 2:** The RADAR analysis technique can be applied to real-world requirements and architecture decision problems.

With respect to Claim 1, one limitation is that RADAR only supports decisions with mutually exclusive options (corresponding to XOR links in feature diagrams); it does not support decisions where multiple options can be selected at the same time (corresponding to OR links in feature diagrams). As a consequence, RADAR is not currently applicable to the problem of requirements (or features) selection that has been studied extensively in search-based requirements engineering [10], [40]. In future work, we intend to extend the language to support decisions with non-mutually exclusive options.

A scalability analysis is presented in a separate report [38]: RADAR's run time and memory usage increase linearly with the size of the design space, which itself increases exponentially with the number of decisions. As a rule of thumb, RADAR exhaustive search strategy can handle problems with up to 10 decisions. Problems with larger design spaces will need alternative heuristic search strategies [39].

B. Usefulness

RADAR is intended to support a requirements and architecture decision method described in previous work [3]. We hope the brief method overview in Section III will have sparked the reader's interest in the method's usefulness.

As an illustration of the method's output, Fig. 7 shows the analysis results for our fraud detection problem. The first

Application	Objectives	Decisions	Variables	Parameters	Design Space	Run-time (sec)
Fraud detection	2	4	31	19	24	0
Security policy	2	2	23	11	6	0
Emergency response	2	10	117	137	6912	111
Satellite image processing	2	10	75	11	1024	3

TABLE I
RADAR APPLICATIONS: PROBLEM SIZES AND ANALYSIS RUN-TIMES

blocking policy	processing type	fraud detection method	alert threshold	FraudDetectionBenefit	InvestigationLoad
block first	continuous	classifier	low	414087	232479
block first	continuous	rule-based		402799	82709
block first	continuous	classifier	medium	402516	52139
block first	continuous	classifier	high	387394	22467

Fig. 7. Optimisation analysis results for the fraud detection model

optimisation objective is to maximise the expected benefit of the fraud detection system where benefit is reduction in financial loss due to fraud compared to the baseline set by the current system. The second optimisation objective is to minimise the fraud investigation load which here is defined as the 95^{th} percentile of the number of alerts:

Objective Max FraudDetectionBenefit = **EV**(Benefit);
Objective Min InvestigationLoad = **percentile**(NbrAlerts, 95);
Benefit = FinancialLossBaseLine − FinancialLoss;

The percentile means that 95 days out of 100, the number of alerts will be below the investigation load.

The optimisation analysis results in Fig. 7 show that all shortlisted solutions include the "block first" policy and "continuous" processing type. This means that, in our model, these two options outperform the "investigate first" policy and "batch" processing on both objectives. But once these two options are selected, the shortlist includes all possible combinations of fraud detection methods and alert thresholds; each combination representing a different tradeoffs between maximising fraud detection benefit and minimising investigation load. To visualise such tradeoffs, RADAR generates the graph in Fig. 8 plotting the objective values for the four shortlised solutions (shown as squares at the top of the graph) and all other non shortlisted ones (shown as circles).

The EVTPI for this problem is 220 and EVPPI for AverageFraudValue is 122. All other parameters have EVTPI below 2. This means that in this model, the only parameter worth investigating further before deciding between the shortlisted solutions is average fraud value. Reducing uncertainty about other parameters would bring no value to the decision.

This example illustrates that a RADAR analysis provides useful feedback to system designers about which decisions are better than others in their model, what objective values can be attained with different designs, what tradeoffs can be made between shortlisted solutions, what parameter uncertainty may deserve additional data collection and analysis before making

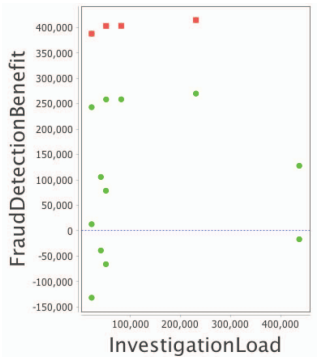

Fig. 8. Pareto Front for the Financial Fraud Detection Prolem

their decision, and what parameter uncertainty does not matter to their decision.

The 'model validity' threat. In our approach, the correctness of the analysis results are relative to the validity of the decision model. If the model's equations are not valid, the predicted objective values for the different solutions might be wrong. In our four examples, although some equations and parameters estimations are based on observed data, we have mostly validated our models by checking that our equations 'make sense' rather than testing them empirically. We can not therefore guarantee their validity.

One should observe that when making decisions about systems that have yet to be built, it will in general not be possible to validate all equations empirically because some of the equations will refer to phenomena that cannot be observed yet. It will only be possible to empirically validate these equations once the system is in use. This is an inherent difficulty of requirements and architecture decision problems.

With respect to the problem of model validation, our approach needs to be compared with the state-of-the-art in requirements and architecture decision making that, by relying on fixed, predefined, and unfalsifiable equations, ignore the issue of model validity. By contrast, RADAR models can be criticised, reviewed, and modified to improve their validity.

Our approach also exposes a gap in requirements and architecture decision making research: we currently lack automated techniques for validating requirements and architecture decision models against observed data, and for automatically calibrating and inferring such models from observed data. Techniques from other fields could be used and adapted [18].

The 'cost of modelling' threat. Another possible problem of our method is that the cost of elaborating the decision models might outweigh its benefits. Our objective in designing RADAR was to reduce the difficulty and cost of modelling compared to existing approaches that require the model to be developed in a general purpose programming language. We have, however, not yet tested how easily people will be able to use our language and tool.

With respect to cost-effectiveness, a potential benefit of our approach is that it enables an iterative modelling and analysis approach where information value analysis might be used to decide what parts of an initially simple model (such as the refactoring model in Section III) should be refined to improve decisions. This will reduce modelling cost by helping modellers develop fine-grained models only where needed and leave other parts of the problem modelled at a coarse level of granularity. We intend to develop and evaluate such iterative approach in future work.

VI. Related Work

RADAR's relations to previous work in requirements and architecture decision making [7], [8], [9], [10], goal-oriented requirements engineering [14], [6], and our previous decision method [3] have been described in Section I. Other quantitative goal modelling techniques also rely on fixed predefined equations and do not analyse uncertainty [41], [42], [43], [44].

In this paper, we have described RADAR as a lightweight stand-alone tool. But in the future, RADAR could also be used to extend existing goal-oriented requirements engineering methods such as KAOS [1], iStar [41] and GRL [44] with a quantitative Bayesian decision analysis layer.

Other quantitative models used in software engineering to reason about uncertainty are Bayesian Belief Networks (BBN) [19] and probabilistic transition systems [45].

A BBN is similar to a RADAR model that would contain no OR-refinements equations. RADAR can thus be viewed as an extension of BBN that include decision points (specified as OR-refinements) to model a set of alternative system designs. RADAR supports multi-objective optimisation and information value analysis not supported by BBN analysis tools. In its most common use, a BBN model specify dependencies between discrete variables using probability tables, whereas RADAR specify dependencies between continuous and Boolean variables using arithmetic and Boolean expressions. BBN tools, however, support more flexible multi-directional belief propagations than RADAR's bottom-up simulation and provides parameter and structure learning techniques not currently supported in RADAR.

Probabilistic transition systems are convenient models to describe and analyse dynamic behaviour of systems exhibiting probabilistic and real-time characteristics. PRISM supports the analysis of several types of probabilistic transition systems [45]. EvoChecker extends PRISM with language constructs that, like RADAR OR-refinements, support the specification of design alternatives [46]. Like RADAR optimisation analysis, EvoChecker can then explore the space of alternatives to find a set of Pareto-optimal solutions with respect to optimisation objectives. However, the modelling paradigm of PRISM/EvoChecker and RADAR are entirely different: RADAR models are declarative equations best suited for modelling stakeholders' objectives, PRISM/EvoChecker models are probabilistic transition systems best suited for modelling dynamic behaviours. The two paradigms are complementary: the result of the analysis of one or more probabilistic transition systems can provide parameter estimations used as input for a RADAR analysis. But RADAR can also be used before the system can be described as a state transition system (for examples, as in the architecture refactoring problem in Section II). In such situation, RADAR allows architects to model decisions using familiar equations without having to elaborate more complex probabilistic transition systems. RADAR's information value analysis can then be used to inform which parts of the system, if any, require more detailed modelling.

VII. Conclusion

Many requirements and architecture decision problems deal with uncertainty and conflicting stakeholders' objectives. Simulating and optimising decisions using quantitative models is a promising approach to support such decisions but the difficulty of elaborating the model limits the adoption of such approach.

We have presented a new modelling language and analysis tool, called RADAR, intended to facilitate the elaboration and analysis of quantitative requirements and architecture decision models. We have shown RADAR is applicable to real-world problems, suggested it produces useful analysis results, and argued that it improves model readability and lowers modelling cost compared to decision models developed in a general purpose programming language.

Unlike state-of-the-art requirements and architecture decision methods that rely on fixed, predetermined equations, RADAR models can be reviewed and modified by decision makers. RADAR's analysis results, however, are only as good as the decision model they are derived from. Future research in requirements and architecture decision making should investigate techniques to support model validation with respect to observed data, and model inference and calibration from observed data.

Acknowledgment

David Stefan implemented RADAR's method for computing the expected value of partial perfect information [25]. Saheed Busari's research is supported by the The Petroleum Technology Development Fund (PTDF) in Nigeria.

REFERENCES

[1] A. van Lamsweerde, *Requirements Engineering: From System Goals to UML Models to Software Specifications*. Wiley, 2009.

[2] N. Rozanski and E. Woods, *Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives*. Addison Wesley, 2011.

[3] E. Letier, D. Stefan, and E. T. Barr, "Uncertainty, risk, and information value in software requirements and architecture," in *36th International Conference on Software Engineering (ICSE 2014)*, 2014, pp. 883–894.

[4] D. Kahneman, *Thinking, Fast and Slow*. Macmillan, 2011.

[5] J. Aranda and S. Easterbrook, "Anchoring and adjustment in software estimation," in *10th European Software Engineering Conference and 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering (ESEC/FSE 2013)*. ACM, 2005, pp. 346–355.

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

[7] G. Ruhe, *Product Release Planning: Methods, Tools and Applications*. CRC Press, 2010.

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

[9] 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)*, 2003, pp. 557–562.

[10] A. M. Pitangueira, R. S. P. Maciel, M. de Oliveira Barros, and A. S. Andrade, "A systematic review of software requirements selection and prioritization using SBSE approaches," in *5th International Symposium on Search Based Software Engineering (SSBSE 2013)*. Springer Berlin Heidelberg, 2013, pp. 188–208.

[11] H. C. Benestad and J. E. Hannay, "A comparison of model-based and judgment-based release planning in incremental software projects," in *33rd International Conference on Software Engineering (ICSE 2011)*, 2011, pp. 766–775.

[12] A. Van Lamsweerde, "Goal-oriented requirements engineering: A guided tour," in *Requirements Engineering, 2001. Proceedings. Fifth IEEE International Symposium on*. IEEE, 2001, pp. 249–262.

[13] E. Letier, "Reasoning about agents in goal-oriented requirements engineering," Ph.D. dissertation, University of Louvain, 2001.

[14] 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)*. ACM, 2004, pp. 53–62.

[15] L. Duboc, E. Letier, D. S. Rosenblum, and T. Wicks, "A case study in eliciting scalability requirements," in *16th IEEE International Requirements Engineering Conference (RE 2008)*, 2008, pp. 247–252.

[16] L. Duboc, E. Letier, and D. S. Rosenblum, "Systematic elaboration of scalability requirements through goal-obstacle analysis," *IEEE Transactions on Software Engineering*, vol. 39, no. 1, pp. 119–140, 2013.

[17] D. J. Hand, C. Whitrow, N. M. Adams, P. Juszczak, and D. Weston, "Performance criteria for plastic card fraud detection tools," *The Journal of the Operational Research Society*, vol. 59, no. 7, pp. 956–962, 2008.

[18] R. L. Winkler, *An introduction to Bayesian inference and decision (2nd Edition)*. Probabilistic Publishing, 2003.

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

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

[21] D. Hubbard, *How to Measure Anything: Finding the Value of Intangibles in Business*. Wiley, 2010.

[22] 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)*. IEEE, 2006, pp. 139–148.

[23] R. Howard, "Information value theory," *IEEE Transactions on Systems Science and Cybernetics*, vol. 2, no. 1, pp. 22–26, 1966.

[24] R. A. Howard, *Readings on the Principles and Applications of Decision Analysis*. Strategic Decisions Group, 1983, vol. 1.

[25] 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.

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

[27] A. S. Sayyad, T. Menzies, and H. Ammar, "On the value of user preferences in search-based software engineering: a case study in software product lines," in *35th International Conference on Software Engineering (ICSE 2013)*, 2013, pp. 492–501.

[28] J. Guo, E. Zulkoski, R. Olaechea, D. Rayside, K. Czarnecki, S. Apel, and J. M. Atlee, "Scaling exact multi-objective combinatorial optimization by parallelization," in *29th ACM/IEEE International Conference on Automated Software Engineering (ASE 2014)*, 2014, pp. 409–420.

[29] C. Henard, M. Papadakis, M. Harman, and Y. L. Traon, "Combining multi-objective search and constraint solving for configuring large software product lines," in *37th IEEE International Conference on Software Engineering (ICSE 2015)*, 2015, pp. 517–528.

[30] 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*, vol. 1917, pp. 849–858, 2000.

[31] T. Parr, *The definitive ANTLR 4 reference*. Pragmatic Bookshelf, 2013.

[32] J. Ellson, E. Gansner, L. Koutsofios, S. C. North, and G. Woodhull, "Graphviz – open source graph drawing tools," in *International Symposium on Graph Drawing*. Springer, 2001, pp. 483–484.

[33] T. Caulfield and D. Pym, "Improving security policy decisions with models," *IEEE Security and Privacy Magazine*, vol. 13, no. 5, pp. 34–41, 2015.

[34] ——, "Modelling and simulating systems security policy," in *8th International Conference on Simulation Tools and Techniques*, 2015, pp. 9–18.

[35] S. Malek, M. Mikic-Rakic, and N. Medvidovic, "A style-aware architectural middleware for resource-constrained, distributed systems," *IEEE Transactions on Software Engineering*, vol. 31, no. 3, pp. 256–272, 2005.

[36] N. Esfahani, S. Malek, and K. Razavi, "Guidearch: guiding the exploration of architectural solution space under uncertainty," in *35th International Conference on Software Engineering (ICSE 2013)*. IEEE, 2013, pp. 43–52.

[37] R. Kazman, J. Asundi, and M. Klien, "Making architecture design decisions: An economic approach," Carnegie Mellon University. Software Engineering Institute, Tech. Rep. CMU/SEI-2002-TR-035, 2002.

[38] S. A. Busari and E. Letier, "Scalability analysis of the RADAR decision support tool." [Online]. Available: https://arxiv.org/abs/1702.02977

[39] M. Harman, S. A. Mansouri, and Y. Zhang, "Search-based software engineering: Trends, techniques and applications," *ACM Computing Surveys (CSUR)*, vol. 45, no. 1, 2012.

[40] Y. Zhang, "Multi-objective search-based requirements selection and optimisation," Ph.D. dissertation, University of London, 2010.

[41] P. Giorgini, J. Mylopoulos, E. Nicchiarelli, and R. Sebastiani, "Reasoning with goal models," in *Conceptual Modeling – ER 2002*. Springer, 2002, pp. 167–181.

[42] M. S. Feather and S. L. Cornford, "Quantitative risk-based requirements reasoning," *Requirements Engineering*, vol. 8, no. 4, pp. 248–265, 2003.

[43] A. van Lamsweerde, "Reasoning about alternative requirements options," in *Conceptual Modeling: Foundations and Applications*. Springer, 2009, pp. 380–397.

[44] 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*, vol. 25, no. 8, pp. 841–877, 2010.

[45] M. Kwiatkowska, G. Norman, and D. Parker, "PRISM 4.0: Verification of probabilistic real-time systems," in *International Conference on Computer Aided Verification (CAV 2011)*. Springer Berlin Heidelberg, 2011, pp. 585–591.

[46] S. Gerasimou, G. Tamburrelli, and R. Calinescu, "Search-based synthesis of probabilistic models for quality-of-service software engineering," in *30th IEEE/ACM International Conference on Automated Software Engineering (ASE 2015)*. IEEE, 2015, pp. 319–330.